Analytical Exploration and Quantification of Nanowire-based Reconfigurable Digital Circuits

THESIS

to obtain the academic degree
Doktoringenieur (Dr.-Ing.)

at the Faculty of Computer Science
of the Technical University of Dresden
submitted

by
Michael Raitza
born on the 23 October 1983 in Rostock, Germany

Promoter:

Prof. Dr. Akash Kumar
Technical University of Dresden

Prof. Dr.-Ing. Thomas Mikolajick
Technical University of Dresden

Date of defense:

20 October 2022

Analytical Exploration and Quantification of Nanowire-based Reconfigurable Digital Circuits

Michael Raitza

Abstract

Integrated circuit development is an industry-driven high-risk high-stakes environment. The time from the concept of a new transistor technology to the market-ready product is measured in decades rather than months or years. This increases the risk for any company endeavouring on the journey of driving a new concept. Additionally to the return on investment being in the far future, it is only to be expected at all in high volume production, increasing the upfront investment. What makes the undertaking worthwhile are the exceptional gains that are to be expected, when the production reaches the market and enables better products. For these reasons, the adoption of new transistor technologies is usually based on small increments with foreseeable impact on the production process. Emerging semiconductor device development must be able to prove its value to its customers, the chip-producing industry, the earlier the better. With this thesis, I provide a new approach for early evaluation of emerging reconfigurable transistors in reconfigurable digital circuits. Reconfigurable transistors are a type of metal-oxide semiconductor field-effect transistor (MOSFET) that features a controllable conduction polarity, i. e. they can be configured by other input signals to work as PMOS or NMOS devices.

Early device and circuit characterisation poses some challenges that are currently largely neglected by the development community. Firstly, to drive transistor development into the right direction, early feedback is necessary, which requires a method that can provide quantitative and qualitative results over a variety of circuit designs and must run mostly automatic. It should also require as little expert knowledge as possible to enable early experimentation on the device and new circuit designs together. Secondly, to actually *run* early, its device model should need as little data as possible to provide meaningful results. The proposed approach of this thesis tackles both challenges and employs model checking, a formal method, to provide a framework for the automated quantitative and qualitative analysis. It pairs a simple transistor device model with a charge transport model of the electrical network.

In this thesis, I establish the notion of transistor-level reconfiguration and show the kinds of reconfigurable standard cell designs the device facilitates. Early investigation resulted in the discovery of certain modes of reconfiguration that the transistor features and their application to design reconfigurable standard cells.

Experiments with device parameters and the design of improved combinational circuits that integrate new reconfigurable standard cells further highlight the need for a thorough investigation and quantification of the new devices and newly available standard cells. As their performance improvements are inconclusive when compared to established CMOS technology, a design space exploration of the possible reconfigurable standard cell variants and a context-aware quantitative analysis turns out to be required.

I show that a charge transport model of the analogue transistor circuit provides the necessary abstraction, precision and compatibility with an automated analysis. Formalised in a domain-specific language (DSL), it enables designers to freely characterise and combine parametrised transistor models, circuit descriptions that are device independent, and re-usable experiment setups that enable the analysis of large families of circuit variants. The language is paired with a design space exploration algorithm that explores all implementation variants of a Boolean function that employs various degrees and modes of reconfiguration. The precision of the device models and circuit performance calculations is validated against state-of-the-art finite element method (FEM) and SPICE simulations of production transistors.

Lastly, I show that the exploration and analysis can be done efficiently using two important Boolean functions. The analysis ranges from worst-case measures, like delay, power dissipation and energy consumption to the detection and quantification of output hazards and the verification of the functionality of a circuit implementation. It ends in presenting average performance results that depend on the statistical characterisation of application scenarios. This makes the approach particularly interesting for measures like energy consumption, where average results are more interesting, and for asynchronous circuit designs which highly depend on average delay performance. I perform the quantitative analysis under various input and output load conditions in over 900 fully automated experiments. It shows that the complexity of the results warrants an extension to electronic design automation flows to fully exploit the capabilities of reconfigurable standard cells. The high degree of automation enables a researcher to use as little as a Boolean function of interest, a transistor model and a set of experiment conditions and queries to perform a wide range quantitative analyses and acquire early results.

Acknowledgements

This thesis has multiple fathers. I want to express my deepest thanks and respect to Hermann Härtig, who took me into his chair when my early aspirations to get promoted almost came to a complete halt after the departure of my first academic mentor Christian Hochberger to a more fruitful place to do research. Hermann and his colleague Marcus Völp, who became my first supervisor of this thesis, took it upon themselves to integrate a computer engineering researcher into the community of systems architecture research. To their effort and endurance with my very different approach to the research topic I owe the opportunity to write this thesis.

After it became apparent that I would remain a computer engineer, Hermann found a new academic home for me in the newly established Processor Design Chair of Akash Kumar, to continue my research under his guidance. With Akash as my doctoral supervisor, my thesis found its final direction. To him I also want to express my deepest thanks and my respect for his leadership.

This thesis is also the product of a willingness and openness to do interdisciplinary research that was facilitated by the Center for Advancing Electronics Dresden (cfaed). I especially would like to thank Thomas Mikolajick and Walter Weber from the semiconductor research community for their financial and scientific support. The cfaed brought me together with Jens Trommer, also from semiconductor research, to whom I owe my gratitude and most of my understanding (and all my FEM simulation data) of reconfigurable transistors. It also set me up with Steffen Märcker from the theoretical computer science community, which spun off a fruitful and intense collaboration that found its expression in the use of model checking in this thesis.

My biggest thanks go to my family who always support me and provide the protected room for development and thought. Dear Uwe, Barbara, dear Thomas, thank you for having you. Last but most importantly, I want to thank my wife Sara, the love of my life, for her love, active support and endurance with a scientist's odd work schedule. You provide the soil on which my creativity can grow its products.

Contents

1 Introduction **1**
- 1.1 Emerging Reconfigurable Transistor Technology . . . 2
- 1.2 Testing and Standard Cell Characterisation . . . 5
- 1.3 Research Questions . . . 6
- 1.4 Design Space Exploration and Quantitative Analysis . . . 7
- 1.5 Contribution . . . 9

2 Fundamental Reconfigurable Circuits **13**
- 2.1 Reconfiguration Redefined . . . 14
  - 2.1.1 Common Understanding of Reconfiguration . . . 14
  - 2.1.2 Reconfiguration is Computation . . . 18
- 2.2 Reconfigurable Transistor . . . 20
  - 2.2.1 Device geometry . . . 20
  - 2.2.2 Electrical properties . . . 23
- 2.3 Fundamental Circuits . . . 28

3 Combinational Circuits and Higher-Order Functions **35**
- 3.1 Programmable Logic Cells . . . 35
  - 3.1.1 Critical Path Delay Estimation using Logical Effort Method 36
  - 3.1.2 Multi-Functional Circuits . . . 40
- 3.2 Improved Conditional Carry Adder . . . 45

4 Constructive DSE for Standard Cells Using MC **51**
- 4.1 Principle Operation of Model Checking . . . 54
  - 4.1.1 Model Types . . . 55
  - 4.1.2 Query Types . . . 58
- 4.2 Overview and Workflow . . . 59
  - 4.2.1 Experiment setup . . . 60
  - 4.2.2 Quantitative Analysis and Results . . . 64
- 4.3 Transistor Circuit Model . . . 65
  - 4.3.1 Direct Logic Network Model . . . 65
  - 4.3.2 Charge Transport Network Model . . . 68

4.3.3 Transistor Model . . . 77
4.3.4 Queries for Quantitative Analysis . . . 88
4.4 Circuit Variant Generation . . . 91
4.4.1 Function Expansion . . . 92

5 Quantitative Analysis of Standard Cells **95**
5.1 Analysis of 3-Input Minority Logic Gate . . . 97
5.1.1 Circuit Variants . . . 97
5.1.2 Worst-Case Analysis . . . 100
5.2 Analysis of 3-Input Exclusive OR Gate . . . 108
5.2.1 Worst-Case Analysis . . . 112
5.2.2 Functional Verification . . . 116
5.2.3 Probabilistic Analysis . . . 120

6 Conclusion and Future Work **125**
6.1 Future Work . . . 127

A Notational conventions **129**

B `prism-gen` Programming Interfaces **131**

Bibliography **133**

Terms & Abbreviations **141**

Chapter 1

Introduction

The semiconductor industry is known to be a high-risk high-stakes environment. New production lines take a tremendous amount of money, time and effort to be established and to start returning their investment. The time scale is measured in decades rather than years or months, as the device that is used in this work a germanium nanowire reconfigurable Schottky junction field effect transistor with multiple independent control gates was first shown in 2006 in [62] and is yet to be picked up for production. As a consequence, production alterations are implemented with great caution and very conservatively, favouring small-step evolution over revolutionary technological jumps. Emerging devices must prove their worth before they can expect to be accepted by the industry.

This is where the current device and circuit characterisation techniques are not sufficient. Current methods expect a technology readiness level of 4–5 ([55]), which is about to be reached now for the mentioned germanium nanowire technology, before significant device models can be constructed, namely SPICE compact models. These models are needed in order to perform sufficiently precise logic circuit performance projections that garner interest of the producing industry or steer further device optimisation. Additionally, producing smaller device structures proves harder to yield enough functioning chips. Thus, the old rule that complexity and reliability are at odds with each other still holds true for chip manufacturing. So, either increasing the functionality per transistor and per standard cell or reducing the size and complexity of a fixed standard cell are still beneficial design goals for emerging technologies. This is what polarity-controllable transistors and reconfigurable circuits are promising candidates for. A design space exploration and quantitative analysis method that works from an earlier technology readiness level would be a valuable contribution to these demands, especially if it focuses on high automation and a decent coverage of the possible circuit implementation variants.

Formal methods, like model checking and theorem proving, provide valuable properties that align especially well with the demands in hardware development,

surveyed in [17] and exemplified in [19]. Both disciplines usually require exact and exhaustive answers to critical questions, which sets them apart from the best-effort computing domain. Both also struggle with the inevitable state space explosion problem which leads hardware development to accept suboptimal circuit design results in exchange for functional correctness guarantees. It also has led to the general acceptance of simulation methods in projecting and validating circuit performance characteristics. In this thesis, I present a design space exploration and quantitative analysis approach that is tailored to probabilistic model checking, a formal method and suitable to investigate logic gates of the size of standard cells. It builds on the model checker PRISM and provides its own input language that is tailored to describe transistors, standard cells and experiment setups to characterise them. Model checking [3] provides the formal framework to exactly model and compute the analogue transistor network and also provides the query language(s), based on temporal logics such as computation tree logic (CTL) and linear time logic (LTL), to ask for interesting quantitative and qualitative circuit properties. It enables the direct computation of extremal values without the need for particular knowledge about the circuit under test. So, instead of needing expert knowledge to provide the right set of simulation input stimuli, the model checker can be directly asked for the worst-case delay or power dissipation for an arbitrary logic gate. After delivering the output, model checking also provides the state transitions as witnesses that allow a close inspection of the circuit behaviour in interesting situations after the fact. What is very valuable, though, for future integration, is that probabilistic model checking is able to determine the long-run average performance characteristics of a circuit. Obviously, the average performance is what is exhibited most of the time by a circuit and, thus, is where optimisations lead to the highest benefits. Especially for measures like energy consumption, that can easily tolerate short but large deviations, optimising for the average may be much more beneficial than optimising for the worst-case. Results, such as the long-run average, can also be computed as rational functions in variables that abstract model characteristics as probabilities, such as the switching frequency of an input.

1.1 Emerging Reconfigurable Transistor Technology

The analysis method that I am going to lay out in this thesis has been conceived to solve the problem of early device characterisation especially for reconfigurable transistor technology that itself is founded in the ambipolar behaviour of each device. *Ambipolarity* describes the property of a semiconductor to conduct both charge carriers, electrons (e^-) and holes (h^+). This is usually a property that designers seek to avoid, because it leads to undesirable short-circuit currents in digital circuits that increase the energy consumption or conflict with the circuit function altogether. In *reconfigurable transistors*, engineers have found ways to control the ambipolar device behaviour electrostatically as shown in Figure 1.1.

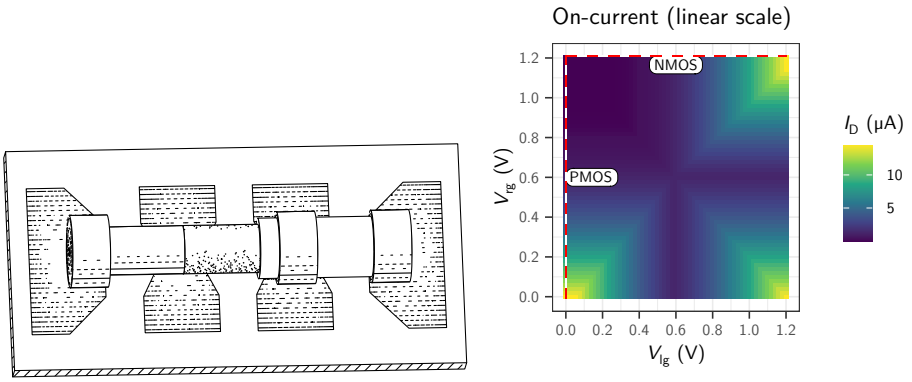


Figure 1.1: Drawing and I–V heat map diagram of a single germanium nanowire RFET from a formal model. 24 nm feature size, comparable to Figure 2.6 on page 26.

The transistor now possesses two gates instead of one. The gates-all-around structure positions each transistor gate at the interface between the metallic wire (dashed), reaching in from the outer contacts, and the semiconductor (stippled). The device can be configured to act as a p-channel field-effect transistor (PFET) or an n-channel field-effect transistor (NFET). By adding another control input, the device polarity can be controlled, in situ, without the need of implanting ions that govern channel polarity like in common CMOS transistors. That is why in this thesis, electrostatically controllable transistors will be referred to as *polarity-controllable transistors*. Transistor polarity control is not completely independent of the rest of the device functionality, which means that it only works if all device terminals are charged to certain voltages relative to each other. Voltage changes to the polarity control gate also change whether the channel opens or closes, unless the second control gate is worked in unison with the first. Additionally, some combinations of transistor contact voltages may leave the device in an uncontrolled ambipolar state. Thus, care must be taken to avoid these states or to let a circuit switch as quickly as possible. The heat map in Figure 1.1 shows the drain current for all possible gate voltages and highlights the normal operation modes. While they touch three corners of the diagram, the fourth corner (lower right) shows a fully-developed ambipolar device. A detailed description of the device's functionality is given in Section 2.2.2, pp. 23 ff.

A single device is used to implement both necessary transistor types that make a complementary circuit, and it can change its polarity through an additional input signal. This gives rise to a type of *reconfigurable circuits* that are founded in Shannon decomposition of Boolean functions. These kinds of circuits are complementary and, thus, consist of a pull-up network and a complementary pull-down network of transistors. What makes them special is that they have a working and

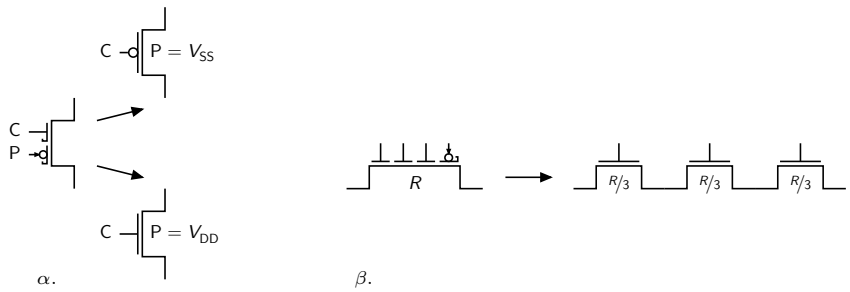


Figure 1.2: α. Polarity control via second transistor gate. β. Reduced virtual channel resistance compared to a series of CMOS devices by using multiple independent gate FETs.

meaningful implementation regardless of which of the transistor networks functions as the pull-up or pull-down network, as long as they are configured to each others opposite.

The technology that I dominantly use in this thesis was first shown in [62], expanded into full reconfigurable FETs in [23] and extended with multiple independent control gates in [56] as shown in Figure 1.2 α and β. It is that of a symmetric nanowire polarity-controllable FET, with emphasis put on its p- and n-channel with (nearly) equal conductance properties. While other concepts of circuit reconfiguration work on an architectural level that sits on top of ready-made CMOS standard cells, *transistor-level reconfiguration* acts on the single device inside reconfigurable standard cells. This means, its effect is not only on the logic level but on the analogue electrical level, as well. It is a well-known fact that due to technological limits and architectural choices, PFETs and NFETs are typically individually sized according to their electrical contribution inside a logic gate. Most famous is the delay performance discrepancy of the NAND and NOR standard cells of older CMOS technology nodes. It gives preference to the NAND circuit, because designing the pull-up and the pull-down networks such that they output the same currents results in a structurally and actually smaller and faster design than a balanced NOR circuit of the same technology. Although this is not unanimously true anymore, it serves as the guiding principle that underlines the importance of a symmetric channel conductance in polarity-controllable devices that are used to implement the reconfigurable circuits that are the subject of this thesis. In addition to this observation, reconfigurable standard cell design relies heavily on a property called *reduced virtual channel resistance* that comes with multiple-independent gate devices in small feature sizes (see Figure 1.2 β). The property enables the implementation of wired-AND or series paths that have the same channel current as a single device. They are the second ingredient that ensures that reconfigurable standard cells can develop their full potential.

The polarity-controllable FETs are based on germanium nanowire heterostructures in a gate-all-around design and implement Schottky junction devices that

are both controllable and polarity-controllable in the same voltage domain. Thus, an output of one logic gate can serve as either an input or a reconfiguration signal of the next logic gate. This is the third important device property, that enables reconfigurable standard cells to work with in-band reconfiguration signals. It sets them apart from similar transistor technology like floating-gate devices used in non-volatile memory or single-transistor sensors that must usually work with separate voltage domains for distinct operations, i. e. reading/writing or excitation/detection.

Chapter 2 establishes the detailed operation of the polarity-controllable transistor and its relation to reconfigurable circuit design. All of these observations and conclusions are results from early experiments on this subject and while we have seen more reconfigurable transistor device design in the last years, these three properties do not receive special attention from semiconductor research. I suppose, in part, this is due to a lack of feedback from the circuit design community. These devices cannot be easily tested without proper models, that are hard to come by in the early design stages, especially because device modelling is usually a complicated process that needs a lot of data (which is not available in the early stages). So, the lack of device data leads to a lack of testing and experimentation which, in turn, leads to a lack of feedback into the device community. This was one motivation to devise the automated design space exploration and quantitative analysis method that I propose in this thesis.

1.2 Testing and Standard Cell Characterisation

Standard cell characterisation for a new device technology is usually conducted by waiting until the development reaches technology readiness level 4 and a SPICE compact model is created. It can then be used by current tool chains to quantify known standard cell implementations, and new standard cell designs can be drafted and tested in tight loops. Considering early device characterisation, recent works, like [50], extract the important device characteristics directly from FEM simulations and apply curve fitting to increase the accuracy. This results in a table model which is, then, compared to and validated against known technology nodes of comparable sizes. A change in device characteristics requires the construction of a new table model, which requires a lengthy series of FEM simulations. In addition, the table model is usually restricted to a limited set of parameters which represents the "normal conditions" the device is used in. Although the device parameters were extracted from single-device simulations, the whole process works on the abstraction level of standard cells rather than transistor circuits. This means that some intricacies of the transient switching behaviour that can have a major impact on circuit performance may get lost due to abstraction. Additionally, this level of abstraction does not cover the numerous circuit implementation variants that reconfigurable standard cells exhibit and that are shown to have significantly diverging quantitative and qualitative characteristics.

Standard cell characterisation methods that are based on simulations also require a lot of expert knowledge to deliver correct and meaningful results. As with all testing schemes, they can only validate known answers or contradict wrong assumptions. Finding the slowest transient of a standard cell requires the tester to ask for the specific input stimulus that exhibits the transient in question. The system cannot be asked for the slowest transient directly and, thus, cannot qualify a given transient as the slowest.

Formal methods can deliver stronger answers and they have been successfully applied to hardware verification on various levels of abstraction on whole systems [29, 31], in logic synthesis [20] and also on the register transfer level (RTL) by augmenting hardware description languages [7, 19]. Static timing analysis on the logic gate level and the verification of RTL designs using probabilistic model checking [30] or with timed automata [1] have proven the general applicability and tractability of formulating hardware verification problems as formal models. Probabilistic model checking [3] in particular provided the necessary tools to successfully apply formal methods to the research problems in this thesis. By describing the standard cells as, basically, analogue circuits that operate on voltages and currents, I could formulate the necessary queries as independent entities that are able to capture the transient switching behaviour of arbitrary logic gates. Queries are formalisations of measures and properties which can be automatically computed and verified by a model checker. This means that I can ask for quantities like worst-case delay, maximum power dissipation, or Boolean properties like the existence of spurious output hazards directly and get an answer as a numerical result or Boolean with a witness/counterexample. It also means that the query to ask for the worst-case delay is independent of the particular circuit under test. Any circuit of a particular size can be checked against it and the model checker identifies the correct transition and quantifies the result automatically.

1.3 Research Questions

Emerging reconfigurable nanotechnology provides exciting new features and is host to a new kind of reconfigurable standard cell design. Yet, the technology still awaits adoption by both the circuit community, which needs assurance that the technology is ready to be produced and the semiconductor producers that need assurance that there are applications for the new technology. Based on this observation, I aim to address the following overall research goal:

Explore the design of transistor-level reconfigurable standard cells and their effects in combinational circuits. Create an automated approach for early technology evaluation to explore the design space of reconfigurable standard cells and to provide quantitative and qualitative results on key metrics.

Today's circuit and technology evaluation tools are centred around the gradual

evolution of CMOS technology that does not aim to provide early feedback to semiconductor research but that accepts new device parameters as a given and optimises standard cell designs around them. With emerging devices, this becomes a relevant aspect which needs to be addressed in the characterisation workflow. Hence, the overall research goal can be broken down into the following individual questions:

1. What is the design space of transistor-level reconfigurable circuits and what kinds of circuit structures and Boolean functions do they support?
2. How do reconfigurable standard cells perform in larger circuit designs? How does their performance change with their surrounding circuitry?
3. Are simple formal device and circuit models feasible and precise enough to allow for both an automated evaluation and valuable early results?
4. Can an automatic quantitative analysis of formal standard cell models be formulated as a tractable problem?
5. Can formal modelling and quantitative characterisation be abstracted in such a way that the needed expert knowledge outside electronic circuit design is minimal?

By these research questions, the centre of this thesis revolves around the question whether a formal analysis can provide the expected answers while maintaining enough precision despite its limited scalability that comes with uncontrollable model state space growth.

1.4 Design Space Exploration and Quantitative Analysis

In this thesis, I develop and present the workflow shown in Figure 1.3. Starting with a Boolean function, a set of queries and a set of experiment conditions on the left, the design space exploration algorithm generates all circuit implementations that adhere to certain criteria that are laid out in Section 4.4.1. These implementations are minimal in the number of transistors, which is why the exploration tool is named `minimal-circuits`. All implementations are generated in the netlist-like language `prism-gen`, that I developed for this thesis. It addresses three concerns and represents them in a single domain-specific language (DSL), a) the parametrisation of transistor devices, b) the description of transistor circuits in a netlist and c) the description and parametrisation of a test fixture. Chapter 4 goes into the depth of describing the inner workings of the DSL. Its main purpose is to provide useful interfaces between the three concerns such that the circuits under test can be constructed independent from the exact transistor being employed. Also, the test fixtures, which are bound to the queries that shall be answered with them, are usually the same for all circuit implementations of

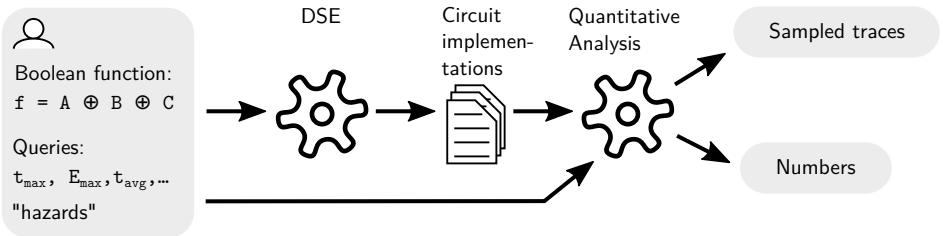


Figure 1.3: Design space exploration and analysis workflow for reconfigurable standard cells.

a single reconfigurable standard cell. So, they too should be independent of the particular circuit implementation. Thus, `prism-gen` enables a designer to analyse multiple circuits in different test fixtures using various transistor devices without the need to rewriting undue amounts of code. The transistor device models are independent of the circuits and are delivered as library elements. Yet, they are still accessible in `prism-gen` such that they can be scaled or even parametrised (e. g. see the drain current imbalance analysis in Figure 2.3 on page 22).

The output of the `prism-gen` compiler is a formal model, more precisely a Markovian model, of the whole experiment that can be processed by the model checker PRISM. The formal model encodes the electrical network as a bipartite graph as can be seen in Figure 1.4 in the example of a NAND circuit that is transformed from the netlist description into a charge transport model. The graph contains the circuit structure and transistors (as charge transport nodes) as well as an input automaton which limits the time and value relations of the circuit input stimuli. One example of an input automaton is shown at the right of Figure 1.4. This automaton describes all possible state transitions of the circuit inputs and contains additional restrictions as to when the next set of transitions is allowed to occur. It usually ensures that the circuit output has stabilised before the next input stimulus is triggered. So, by describing a rule by which the input stimuli may develop, instead of a fixed trace of input stimuli at fixed intervals, the model checker is able to explore all relevant input transitions by itself. Queries that assess worst-case properties work on non-deterministic automata while queries that implement statistical application behaviour work on probabilistic automata. This is reflected in two separate test fixtures with two different input automata.

The last input into the DSE and analysis workflow are the queries, which correlate with their experiment models. They are also largely independent of the circuits in question and usually only depend on the number of inputs into and outputs out of the experiment. Both the DSE and the quantitative analysis run completely automatic, delivering different kinds of output, depending on the query. Most queries deliver direct numeric results, but for some analyses, it might be worthwhile to look at the transient behaviour of the circuit under test. This is delivered as witness states by the model checker which can be used to generate the

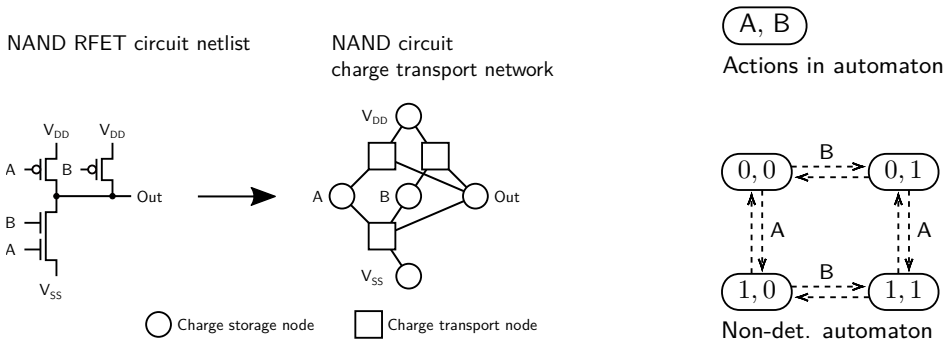


Figure 1.4: *Left* Conversion of a netlist circuit description into the formal model by `prism-gen`. *Right* A typical non-deterministic input automaton with inputs A and B that is compatible to the circuit.

sampled traces in question. Lastly, the model checker is able to return a rational function that can generate the numeric results of a family of input parameters. So, instead of re-running the costly quantitative analysis for a new set of input parameters, the problem can be reduced to computing a simple rational function.

One of my contributions is, that I use this automated workflow to perform a mass analysis of two reconfigurable standard cells which, together, have more than 60 implementations. More than 900 experiments deliver all interesting quantitative and qualitative results that are shown in Chapter 5.

1.5 Contribution

The collaboration with semiconductor research uncovered a demand for early circuit characterisation and device performance projections in standard cells. One additional aspect was to have a device model that is easy enough to be altered when more beneficial characteristics are determined from experiments or when the device production work flow would change the final device characteristics, which make reevaluation necessary. As the semiconductor producing industry is very conservative, strong predictions regarding viable circuit architectures were also sought for as a convincing tool to pick up the new transistor devices.

The contributions of this thesis are the research of fundamental reconfigurable logic gates to show the conceptional foundation of transistor-level reconfiguration and the several reconfiguration types that exist and reconfiguration modes that can be used to drive the circuits. This thesis also shows the feasibility of reconfigurable standard cells in larger combinational circuit designs and their usefulness in implementing higher-order functions, their contribution to the reduction of circuit complexity and performance improvements. Its main contribution is a comprehensive design space exploration and analysis method for reconfigurable

digital circuits that is built on formal methods. It is used to show the automated exploration and comparative analysis of the 3-MIN and 3-XOR functions and their various reconfigurable implementations. Multiple measures are covered in the quantitative analysis for a range of experiment conditions.

Fundamental reconfigurable circuits Reconfigurable circuits that are based on polarity-controllable transistors, like germanium nanowire devices [56], implement a certain kind of reconfiguration that exploits Shannon decomposition of Boolean functions. In Chapter 2, I investigate fundamental circuit implementations and how they make use of polarity-control and the reduced series resistance of multiple-independent gate transistors. Device geometry, functional and performance symmetries turn out to play a major rôle in the effectiveness of the device for a well-performing reconfigurable circuit. I infer three device reconfiguration modes and their influence on circuit design possibilities.

Combinational circuits Reconfigurable implementations of logic gates have to prove their efficiency in larger circuit designs. In Chapter 3, I compare the reconfigurable implementations of basic logic gates to their standard CMOS counterparts implemented in a state-of-the-art device node. Their structural differences are shown by comparing their logical efforts (see [53]) in addition to absolute delay values. Strong variability of results motivate the later analysis of the whole spectrum of reconfigurable implementations of a certain logic gate, as neither the logical effort nor the delay are sufficient to judge the trade-offs between circuit structure and performance. In that chapter, I also show the improvements that reconfigurable standard cells can bring to computation-oriented circuits, by implementing an ALU with higher delay performance and less energy consumption. A complex conditional sum adder serves as the example for how reconfigurable standard cells can improve circuit characteristics if their reconfigurability is not a user-visible function but remains embedded to compress a fixed circuit functionality into a smaller circuit design.

Constructive circuit exploration and analysis The initial studies show that a comprehensive analysis of reconfigurable circuit implementations is necessary to determine their actual performance range and their suitability in the design of combinational circuits. Additionally, reevaluation, performance projections and adaptation to modified transistor characteristics are a main goal for the methodology and tool set that I present in Chapter 4. While hand-picked implementations could show the feasibility, the comparison between the technologies also showed that the actual performance depends on intricate details of the circuit structure and the individual transistor device. Thus, the methodology must allow a researcher to perform an automated exploration and analysis of the reconfigurable implementations of a certain Boolean function. In this thesis, I devise a charge transport network model that operates on simple analytical transistor de-

vice models. It is precise enough to capture the analogue device behaviour and is, yet, simple enough that it can be implemented as a formal model. This model is built and analysed by the probabilistic model checker PRISM which checks device- and circuit-independent queries against it. They can directly answer questions for extremal values like worst-case delay and power dissipation as well as support the detection and enumeration of spurious output hazards and the verification of circuit functionality. The models and the experiment descriptions are implemented in the domain-specific language `prism-gen` that abstracts the model details into a netlist-like description language that enables the composition of complex circuits from simpler ones while maintaining detailed access to single transistor or experiment properties if necessary. To show the validity of the proposed method, I compare the germanium nanowire model against TCAD simulation data from [58, 56]. Additionally, I also implement the CMOS production device from [41] and compare the performance of standard cells implemented in the proposed method against commercially available SPICE models to show the precision of network and device model at a circuit level. The design space exploration of reconfigurable circuit implementations is constructive such that in a given set of constraints, all implementations are considered and are guaranteed to be functionally correct. I also show why timed automata are not a suitable implementation technique despite being an established method and successfully used in hardware verification on the register-transfer level.

Automated quantitative analysis To put the proposed method to the test, I present a comprehensive evaluation of two significant Boolean functions, which shows its range and abilities. In Chapter 5, I show a worst-case analysis of all reconfigurable 3-input minority circuit variants in comparison to each other and the static implementation. The analysis reveals that both input inverter sharing and output load sensitivity must also be considered to get a complete picture of the performance properties of individual implementations as worst-case analyses prove to be too simplistic to capture all realistic use cases. To address computation-intensive scenarios, I also investigate the 3-input exclusive-OR function. This analysis includes hazard detection and shows that the direct access to extremal values uncovers the worst-case states for different measures like delay and energy may deviate from each other, underlining that each measure warrants an independent analysis that is easily achievable with the proposed method. Future technological improvements will have to concentrate on improving the average performance case, because this is where the most optimisation potential lies for best-effort computing. This is especially true for measures like energy consumption, in which short extreme outliers can be easily tolerated. These measures are hard to come by with simulation techniques but become accessible with probabilistic model checking. I show long-run average results for the 3-XOR circuits and parametric delay calculations whose results are directly usable in future EDA tools.

Chapter 2

Fundamental Reconfigurable Circuits

Reconfiguration and reconfigurable hardware has settled in a prominent niche in both research and application since the 1990's when it became powerful enough to serve as an implementation basis for real-world applications. It has become a fixed go-to implementation vehicle for certain high-bandwidth software-defined communications and filtering applications with ever-varying degrees of reconfigurability. Its success hinges on its ability to strike a balance between the system architect's (or user's) uncertainty of the exact computation that the target system needs to perform and strong requirements regarding computational efficiency. Uncertainty requires some degree of programmability which directly conflicts computational efficiency, which is either dictated by necessity in cyber-physical systems or has monetary reasons in best-effort computing applications (faster = more money). Additionally, programmable systems are, at least somewhat, adaptable to new conditions and repairable in the field.

There has been numerous preliminary work on reconfigurable hardware design and architecture. Principle work and how the approach in this thesis differs from it is introduced in Section 2.1.1. The concept of *reconfiguration*, as explained in the next section, is closely tied to the fundamental transistor device that enables the digital circuitry in the first place. It is no architectural concept that sits on top of regular CMOS standard cells. Numerous transistors have been proposed that would be suitable as implementation vehicles for the kind of reconfiguration described in this thesis. Silicon and germanium nanowire transistors that use Schottky junctions are one class of devices and are shown in [62, 23, 56, 51, 11]. They also serve as the implementation devices to showcase the quantitative standard cell characterisations in this thesis, because their development already produced sufficient research data to not only model the devices but to also validate the models against independent simulations.

Dual metal-gate planar FETs are proposed in [28], but carbon nanoribbon devices ([16]) and other 2-D material devices ([44]) also show promising characteristics regarding polarity controllability. For instance, heterostructures from carbon nanotubes and the 2-D material MoS_2 that form a reconfigurable tunnelling transistor are shown in [33].

2.1 Reconfiguration Redefined

The term *reconfiguration* will be used often throughout this thesis and, thus, warrants an explanation of what is understood as reconfiguration and how it is different from common usage throughout the field. Reconfiguration, in this work, is bound to a technological ability of the primary active component, the transistor. The transistors we consider in this work, and which will be detailed in Section 2.2, all have the ability to have their channel polarity electrically controlled. Thus, they can conduct either positive charge carriers or negative charge carriers, controlled by an additional input. In combination with certain circuit topologies, introduced in Section 2.3, polarity control enables us to perform different Boolean functions depending on the *configuration* of the polarity-control inputs of the transistors. The circuit is said to be *reconfigured* from one Boolean function to another.

2.1.1 Common Understanding of Reconfiguration

Circuit design always has been an industry-driven field of research. This means that terms do not always have their foundation in science but may stem from product marketing. When looking closely at the use of the term *reconfiguration* it becomes apparent that it is used more colloquially than in a precise narrow meaning. Programmable array logics (PALs), programmable logic arrays (PLAs) and field-programmable logic arrays (FPGAs), for instance, are clearly considered reconfigurable circuits but are called *programmable*. Their implementations also differ a lot from each other.

The PAL was first described in [5] as an array of programmable fuses. Each input signal is driving two columns of the array, one as direct value and one inverted. Programmable fuses allow the designer to connect each column to select rows, which then connect to (4-input) AND gates. Two AND gates are pair-wise connected to a final OR gate, whose output signal can be used directly or inverted. This means, all logic must be expressed in product terms under consideration of the limitations introduced by the AND and OR logic after the array. Routing and the construction of sequential logic must be done off chip by connecting an output to another input. Configuration strictly considers the description of product terms and output signal inversion. Reconfiguration is not possible in these devices so they are, strictly speaking, not reconfigurable. Nevertheless, they introduce the important concept of a semiconductor fabric whose connections are done in late

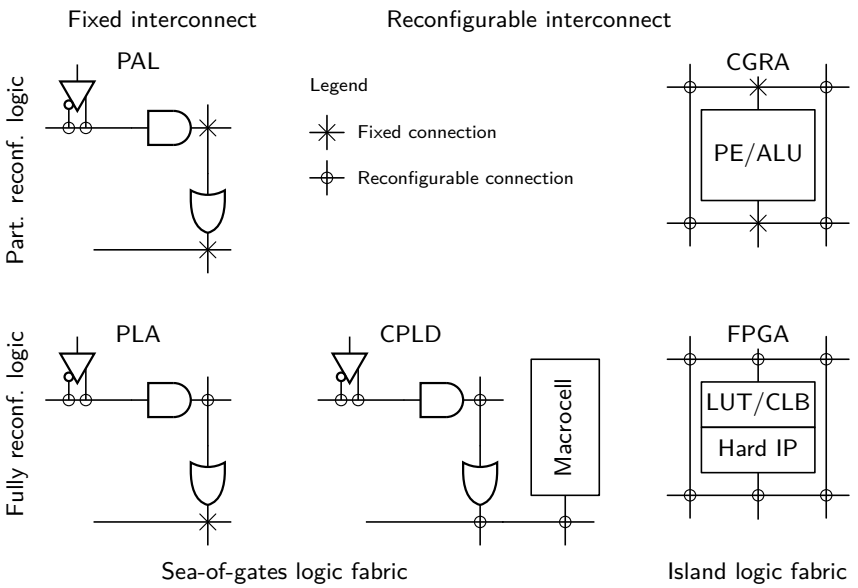


Figure 2.1: Comparison of common reconfigurable architectures along three axes: reconfigurability of the logic fabric, reconfigurability of the interconnect, and the design of the logic fabric.

production depending on the intended application.

The PLA extends this concept, as described in [22], by introducing a second matrix between the AND gates and the OR gates. Again, each AND gate drives two rows, directly and inverted, which can be connected to select columns via fuses. A number of columns, usually eight, drives a single OR gate. This design does not change the fundamental expressivity of its reconfiguration nor does it change the level of architecture it is applied on, but it enables a better compression of the logic functions by enabling the expression of logic in configurable sum-of-products terms. Most notable, product terms that are far away from each other or use radically different inputs can now be shared between several OR gates as these no longer only connect to neighbouring AND gates. The architecture also features internal feedback paths, leading to a more reliable sequential logic, as their impact on signal propagation and latency can now be depended upon.

The deciding technological step forward comes by the use of erasable fuses to implement the configurable connections which are introduced in programmable logic devices (PLDs) and complex programmable logic devices (CPLDs). Though their fundamental fabric design is similar to PLAs, they can be reconfigured in the field. Architecture-wise, CPLDs contribute macrocells (hard macros or hard intellectual property cores (IP cores) in modern terms) in addition to the sea-of-gates logic fabric and the programmable interconnect, which can multiplex

between multiple instance of logic fabric and macrocells. The interconnect allows for deeper logic and carrying inner state by providing internal feedback paths and defined ways to construct flip flops.

FPGAs were introduced in [8] and make an important step forward to the, now common, understanding of reconfiguration by introducing several new reconfiguration primitives. They introduce the 2-dimensional array architecture of programmable logic elements surrounded by programmable routing, which was pioneered in a fixed form by contemporary fabricated gate arrays. They are also the first devices to use static random access memory (SRAM) to capture the device program instead of PROMs or fuses. Although at the time of its inception it was still two decades away from its realisation, this gave rise to the idea of dynamic hardware; hardware that can be reconfigured not only in the field but even in operation, so, while the surrounding hardware still performs its function. Today, this concept is subsumed under the name of dynamic partial reconfiguration or dynamic function exchange. Another important architectural contribution is the use of lookup tables (LUTs) for Boolean function implementation instead of a sea-of-gates design. Implementing the full truth table of a function, which looks excessive on paper, turns out to be the driving factor for a flexible, synthesisable and routable logic design, as it simplifies the natural entanglement of logic placement and routing by featuring constant-delay Boolean functions, as long as they fit into a single LUT. Finally, replacing crossbars with chains of pass transistors enables a flexible 2-dimensional routing. FPGAs introduce reconfiguration on another architectural level, the interconnect. Starting with CPLDs, the programmable interconnect inside an FPGA is highly structured into local connection, longer uninterrupted connections and long lines that can reach across the chip. In addition, FPGAs feature a separate, low-skew clock network with the ability to cross signals from and to either network to achieve maximum flexibility and highest performance. Switch boxes form the programmable part of the interconnect and allow for efficient XY routing across the FPGA's surface. This 2-dimensional network of routes with islands of hard macros and configurable logic blocks in between result in the most powerful and versatile reconfigurable architecture to date, with a 14 times larger market size of 6 billion US dollars [61], than CPLDs, with a market size of 421 million US dollars [49] in 2021.

All this flexibility comes at a high price, though. Computing the logic placement and the routing between the placed logic are both interdependent and NP-hard problems in themselves. Computing even simple designs needs several orders of magnitude more powerful machines than for the next class of reconfigurable architectures, the coarse-grain reconfigurable architecture (CGRA).

CGRAs can be regarded as a set of arithmetic logical units (ALUs) connected by a network on chip (NoC). This means, they can be considered distributed processing systems but are called (coarse-grain) *reconfigurable*. Their strength lies in the focus on efficiently connecting common-case processing elements and can be considered a case of the principle *Convention over Configuration*. By having a convention that defines the interface between the processing elements, i. e. a 64-bit

bus, request-acknowledge signals or a common clock, routing between numerous processing elements becomes a more constrained and, thus, easier task. It also reduces the size of the device program by several orders of magnitude, enabling successive reconfiguration while in operation. Reconfiguration works completely on a high architectural level, specifying the abstract connections between complex processing elements.

This shows that reconfiguration can neither be pinned to a specific mechanism nor a specific architectural level, nor a specific algorithm. Figure 2.1 shows the evolution from PALs to CPLDs, the revolutionary change in architecture in FPGAs and the simplification back to partially reconfigurable logic to gain speed and run-time reconfigurability in CGRAs.

In very general terms, reconfiguration can be described as a mechanism, that allows a designer to modify the functionality of an integrated circuit. The access to this mechanism and the resources it acts upon inside the integrated circuit are not part of the regular logic that integrated circuit uses to drive the target application. So, a reconfigurable circuit always uses some extra resources to govern the interaction between and functionality of the user-visible logic resources.

It is important to note that any implementation of an algorithm conceptually consists of at least three parts, a data path leading from the input to the output, a reducer component destroying entropy (thus, performing a computation), and a control path that steers the behaviour of sub-components. These are the same parts that constitute a Turing machine, the way of data from the band through the head, the head as the mechanism performing mechanical work and the rules that describe what actions the head can perform. While the band is regarded part of the Turing machine, data memory is normally not regarded as being part of an algorithm. Circuits cannot be an exception to this observation.

So, using this mental model to approach reconfiguration of circuits, we arrive at the same concept we traditionally use in computer science, programmability. The control path is the implementation of our program and can, in general, be fed by input on the data path. Like in central processing units (CPUs), the behaviour of the reducer components depends on the input. When an ALU is either computing, say, addition or subtraction, the circuit is (re-)configured to perform either computation. One could argue, that the difference between these two is, that the ALU is always performing both operations and we merely control the propagation of the results or, even if we halt execution of unused units, we still need to have them physically implemented. Whereby in a reconfigurable circuit, the unused logic is not even there. Well, for reconfiguration I will be talking about later in this work, this is not true. There will be no unused logic sitting dark as a consequence of reconfiguration. Also, to be truthful, one must accept that reconfigurable circuits must also keep unused resources around. If we could build a reconfigurable circuit precisely with the amount of logic that we needed for our application, it would not need to be reconfigurable at all but would already be the final application-specific integrated circuit (ASIC). So, neither of the two are valid arguments to treat reconfiguration in FPGAs any different from

programming circuits like CPUs.

So, while it might be useful to distinguish reconfiguration from programming, this distinction has no solid scientific foundation. It is rather a social distinction like differentiating between scripting (as in JavaScript) and programming (as by programming in C). At least, its usefulness highly depends on how narrow the communication context is in which reconfiguration needs to be set apart from computation.

2.1.2 Reconfiguration is Computation

In this thesis, I will use reconfiguration in a restricted meaning that is defined by the polarity-control feature of the transistors that I use. Used in complementary logic circuit designs, it can be captured by the mathematical formalism of self-dual Boolean functions.

Using polarity-controllable transistors, a complementary circuit is reconfigured by changing the sensitivity of a particular transistor towards its inputs, which is equivalent to changing its channel polarity from NMOS to PMOS or vice versa. That is why I call this mode of reconfiguration transistor-level reconfiguration. Mapped to a whole complementary logic circuit, this means that parts or all of the PMOS network is reconfigured to an NMOS network and, at the same time, its complementary part is reconfigured in reverse. For complementary logic circuits, this mode of reconfiguration completely stays in the Boolean domain or the Boolean domain is closed under reconfiguration. The difference to reconfigurable LUTs is that transistors can be individually reconfigured, giving transistor-level reconfiguration greater power to also perform non-Boolean reconfiguration. With few exceptions, which are expressly mentioned when they appear, this work concentrates on constructing, evaluating and showcasing logic circuits that make use of purely Boolean transistor-level reconfiguration.

Now, the mathematical foundation to Boolean transistor-level reconfiguration lies in self-dual Boolean functions.

Definition 1 *A neutral function is a Boolean function that has equal number of min terms and max terms.*

Additionally, self-dual functions must not have mutually exclusive terms because otherwise they could not have a dual term for each other term.

Definition 2 *Two Boolean terms, which are both either conjunctions or disjunctions, are called mutually exclusive iff.*

1. *they use the same variables,*
2. *all their variables are complementary to each other and*
3. *they evaluate to the same output.*

Table 2.1: Example of a Boolean function that is neutral but not self dual. Each term and its dual term is described as is the expected output to make a particular row self-dual to its dual row.

a	b	Out	Term	Self-dual term	Expected Out (by n'th row)	Mutually exclusive term
0	0	1	$\overline{a} \wedge \overline{b}$	$a \vee b$	0 (by 4)	$a \wedge b$
0	1	0	$\overline{a} \vee b$	$a \wedge \overline{b}$	1 (by 3)	$a \vee \overline{b}$
1	0	0	$a \vee \overline{b}$	$\overline{a} \wedge b$	1 (by 2)	$\overline{a} \vee b$
1	1	1	$a \wedge b$	$\overline{a} \vee \overline{b}$	0 (by 1)	$\overline{a} \wedge \overline{b}$

Definition 3 *The dual term B' of a Boolean term B is defined as the term that is constructed from B by replacing all binary and unary mappings with their inverses (e. g. $\vee \leftrightarrow \wedge$, $\circ \leftrightarrow \overline{\circ}$) and exchanging the neutral elements (e. g. $0 \leftrightarrow 1$).*

Let us consider Boolean terms over three variables a, b and c. The mutually exclusive term to $\overline{a \wedge b \wedge c}$, which is $a \wedge b \wedge c$, cannot be equal to the dual term, which is $a \vee b \vee c$. If that would be the case, it would mean that the output of the function that uses these terms does not depend on the three variables, at all.

Definition 4 *A self-dual Boolean function is a Boolean function equal to its own dual, i. e. each of its Boolean terms is replaced by its dual. A Boolean function over n variables is called partially self-dual when it contains a self-dual Boolean function over k variables with $k < n$.*

An in-depth explanation of self-dual functions and their properties can be found in [4].

The Truth table 2.1 for the function over two variables has $2^2 = 4$ lines, some of them minterms some maxterms depending on the function value. And each one is mutually exclusive to one and only one other term as shown in the column Remark. The table also lists the term, its accompanying dual term and, given the dual term's output from the truth table, the expected output to make both terms dual. Only when the output and the expected output match for each line, is the function self dual.

The smallest non-trivial self-dual Boolean function is over three variables and is shown in Table 2.2. While the left half of Table 2.2 lists all possibly self-dual functions over two variables. Closer inspection reveals, though, that none of them depends on both variables. This concludes that there are no self-dual Boolean functions over two variables. It is the 3-input minority function, which has the added benefit of covering the complete Boolean algebra, due to it being usable as an inverter. This function is self-dual in every variable. So, each variable can be used to select between a term and its dual, and it does so for all terms of the function. This property can be directly translated into a mechanism that provides Boolean transistor-level reconfiguration.

Table 2.2: Left: All possibly self-dual dyadic Boolean functions. None actually depend on both variables, proving that there are no self-dual dyadic functions. Right: Example of the smallest non-trivial self-dual function which also covers the complete Boolean algebra, the 3-input minority.

a	b	all self-dual dyadic Boolean functions			
0	0	1	0	0	1
0	1	1	0	1	0
1	0	0	1	0	1
1	1	0	1	1	0
	f	$\overline{a}$	a	$\overline{b}$	b

a	b	c	3-MIN
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Definition 5 *Given a self-dual function f over n variables and the variables be partitioned into a set of reconfiguration variables R and a set of static input variables S.* Boolean transistor-level reconfiguration *maps reconfiguration variables to the polarity-control inputs and source inputs of transistors such that the resulting logic circuit implements the set of dual terms, switching between either the term or its dual depending on the values of the reconfiguration variables.*

This is the definition of reconfiguration that will be used throughout the remainder of this document unless noted otherwise.

2.2 Reconfigurable Transistor

Self-dual Boolean functions and their relation to reconfigurability are directly reflected in the electrical behaviour of the transistors I am going to use in this thesis. The devices used in the remainder of this thesis are nanowire transistors built in a silicon or germanium technology. They were conceived with their basic electrical characteristic in mind, which sets them apart from off-the-shelf MOSFET devices—their ability to change the conductance of their channel between e^- carrier conductance to h^+ carrier conductance with the help of an extra transistor gate. As it turns out, when implementing static CMOS logic gates, this property coincides with Boolean self-duality.

2.2.1 Device geometry

Industrially manufactured nanowire transistors are etched out of the bulk material in a top-down process, comparable to standard CMOS devices. The drawing of an ideal silicon nanowire (SINW) transistor in Figure 2.2 shows a grown device which

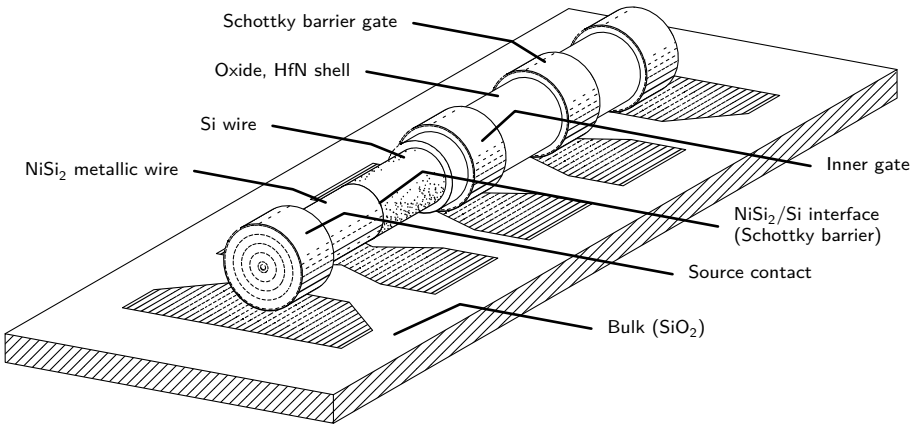


Figure 2.2: Idealised drawing of a grown silicon nanowire transistor. At its core, the silicon wire and metallic $NiSi_2$ wires form a heterostructure with two Schottky barriers. All-around gates over the barriers control the channel polarity and conductivity. Additional inner gates add a wired-AND functionality to the device without impacting device performance.

is functionally the same but whose components are easier to identify and explain. It's proportions were chosen for instructive purposes and are not to scale. All hatched components in Figure 2.2, like the contact surfaces, depict metallic parts, white surfaces are oxides, and stippled objects are semiconductors.

In production, the wire is grown under a gold particle or etched from the bulk material. After formation, the wire is metallised from its ends by intrusion of nickel, forming nickel silicide ($NiSi_2$). Nickel silicide keeps a sharp interface to the silicon wire and its intrusion depth can be controlled by keeping a precisely timed temperature profile or by depositing exact amounts of nickel onto the source and drain contacts and run the intrusion until all material is depleted.

It is important that the $NiSi_2$/Si interface forms directly under either of the outer gate contacts for maximum control over the Schottky barrier and, thus, good device performance (cf. [56] pp. 18). The Schottky barriers form an energy barrier for either types of charge carriers which cannot be overcome by the voltage difference between the source and drain contact. Each Schottky barrier gate can deform the conduction band along the semiconducting middle wire section and depending on the material, the gates need about 0.6 V(Si) or 0.3 V(Ge) difference to the source to allow charge carriers to tunnel through the barrier (the device-intrinsic threshold voltages). For reasonable wire lengths (up to a few hundred nanometres), charge transport inside the semiconducting section is ballistic, which is the reason that the inner gate shown in Figure 2.2 does not increase the channels intrinsic resistance and, thus, does not negatively affect device performance. Inner gates show different performance characteristics than Schottky barrier gates which

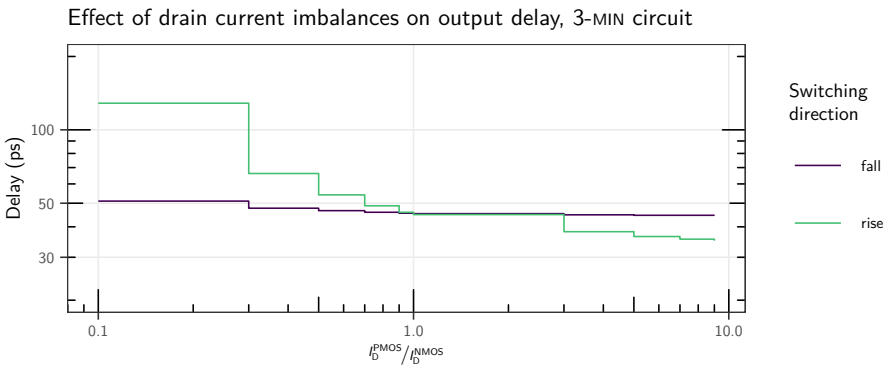


Figure 2.3: Modeling of the effects of an imbalance of drain currents between PMOS and NMOS configurations of germanium MIGFETs. 3-MIN circuit variant 1,bl,sg, output load $H = 1$, worst-case delay.

relate to the physical structure of the device. As they do not influence a Schottky barrier, inner gates have a smaller threshold voltage, thus, need less energy and usually switch faster than their Schottky barrier siblings. It is also possible to have multiple inner gates on a single nanowire and reconfigurable circuits will make use of them.

The geometry of SINW transistors has another benefit that becomes important for building reconfigurable circuits. The amount of material in the hafnium nitride (HfN) shell can be used to parametrise the on-current through the wire. Changing the thickness of the shell changes the pressure on the wire, which influences charge carrier mobility differently for h^+ carriers than e^- carriers. This way, apart from unavoidable production variations, the PMOS and NMOS on-currents can be balanced out enabling the design of CMOS logic gates without severe performance imbalances.

Figure 2.3 shows the effects of drain current imbalances on circuit performance. The graph depicts the worst-case delay of a fully reconfigurable 3-MIN circuit, specificly the variant 1, bl, sg, which is similar to the one shown in Figure 2.10 I, over the quotient $I_{\mathrm{D}}^{\mathrm{PMOS}}/I_{\mathrm{D}}^{\mathrm{NMOS}}$. The nomenclature of circuit variants will be explained further in Chapter 5, but the signifier *bl* stands for a distribution of the reconfigurable signals over all input signals and the *sg* describes the reconfiguration mode the transistors are used in and is explained Table 2.3.

The results were retrieved by the model checking approach that is also explained in full detail in Chapter 4, but they are not measurements or simulation results but exact numbers (within the precision of the model). The curves show the rise and fall delay separately, to distinguish PMOS-driven switching events from NMOS-driven events. As can be seen from Figure 2.3, for this particular circuit, there is a zone between 0.8 and 3 in which the imbalance as almost no

effect on the worst-case output delay. This is well within range of fabrication tolerances, such that current imbalances, although being a factor to consider are not a show stopper to transistor-level reconfiguration.

As shown in Figure 2.2, the device is completely symmetric around its middle point. This is also true for the produced device, thus, all names, such as source and drain contact or program and control gate, can be given to any suitable point of the device and only bear meaning in direct relation to each other or the surroundings of the device but not to physical points of the fabricated transistor. This means that the contact points in Figure 2.2 could be named in reverse.

The example shown in Figure 2.2 features silicon as an example technology mainly, because this was the most advanced nanowire technology available during the research for this thesis. Nonetheless, germanium nickel-based nanowires have been shown in [56] and various other 2-D materials have been demonstrated that follow the same basic concept and have similar properties. So, although, the example may come across as being limited to a particular technology, it was chosen for its simplicity and availability of research data, which was most important for developing circuits and models on top the device itself. I will use silicon and germanium devices, concepts and data, based on the theses of André Heinzig [23] and Jens Trommer [56] interchangeably and where it fits the purpose of proper display. As long as some basic electrical properties are retained, the actual device is not of importance to the applicability of the approaches demonstrated in this thesis. I refer the reader to above theses for an extensive read on the construction of nanowire transistors.

2.2.2 Electrical properties

To make it easier to grasp the transistor characteristics, we will use the following conventions to name the device terminals. In logic gates, the *source contact* will be placed on the input side or on the outer edge of the circuit, where the input signals and voltage supply is connected; the *drain contact* will always point to the output or the middle of the logic gate. This placement is used for both NMOS and PMOS transistors and is unlike the conventional terminology used in CMOS logic gates. These conventions will become important in later chapters concerning the abstraction of the transistor device into a probabilistic model.

The top circuit symbol in Figure 2.4 directly corresponds to the geometrical layout shown in Figure 2.2. Following the drawing style of Schottky devices, the Schottky barrier (SB) gates are drawn with hooks. Unlike other Schottky device symbols, the hooks are only drawn towards the drain side of the device and, following above conventions, will point to the middle axis of a CMOS logic gate. The naming of gates and contacts is no longer arbitrary and the gate with the hook facing towards the contact is always the drain-side gate, which is also the output side. By using these drawing conventions, each transistor gate can be assigned a fixed function out of two.

The bottom drawing in Figure 2.2 shows that the *program gate*, which is

responsible for controlling the channel polarity and thus the PMOS or NMOS characteristic, sits at the drain side. It is always a Schottky barrier gate. The inversion circle marks this gate as being connected to the device's polarity, and the arrow towards it shows that the transistor's polarity is controllable via a voltage applied to the gate. All other transistor gates are called *control gates*, as either can steer the channel open or close, but they do not affect the channel polarity. Control gates never carry an inversion circle to mark PMOS channel polarity, which must be deduced solely from the program gate input.

The term RFET is used for the whole class of reconfigurable transistors and, generally, I will make no distinction between RFETs that employ exactly two transistor gates and three independent gate field-effect transistors (TIGFETs) or MIGFETs. In addition, I will not discriminate polarity-controllable transistors shown in works of the École polytechnique fédérale de Lausanne (EPFL), in which the two outer gates are electrically connected and work in unison. In the circuit drawings throughout this document, transistors may have two or more gates which work independent unless explicitly connected, and whose characteristics are according to the following paragraphs unless mentioned otherwise.

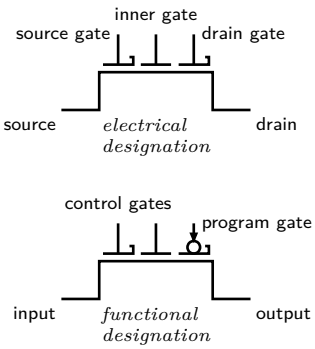


Figure 2.4: RFET terminal designations

Due to the nature of the device, either Schottky barrier prohibits the flow of charge carriers into the channel when the gates are left floating. As the semiconductor erects a band gap which excludes certain energy states of charge carriers within a surrounding electric field, the metal conductors at the outer ends force the charge carriers into states that are well within the band gap. The five band diagrams in Figure 2.5 show the physical expansion of the channel on the abscissa and the energy (as inverse $-E$, so that a higher value translates to higher charged electrons) on the ordinate. The observations, here, are just qualitative and thus no values are supplied in any dimension.

A transistor with its gates left floating (and whose charges meet with the electrical field created between the source and drain contacts) would exhibit the unbent band gap shown in Figure 2.5 in Band Diagram I. The axial expansion of an (unbent) band gap prohibits charge carriers from tunnelling past either Schottky barrier in both directions, thus, closing the transistor for either charge carriers. It is, naturally, an unstable state with no usefulness to digital circuit design. The symbol of the transistor above and the voltages noted at the source/drain contacts reveal that the band diagrams depict the transistor in NMOS configuration. Starting from this outset, the first question is, *how to properly program the channel for e^- charge carrier transport?*

Band Diagram II shows the NMOS-configured and closed transistor. The program gate on the right, charged to 1.2 V, blocks h^+ charge carriers from entering

from the right and raising to the top left. It does not, though, affect the source side of the device. Thus, the first thing to note is, that channel programming works by inhibiting unwanted charge carriers from entering the channel from the "output" or drain side, instead of drawing wanted charge carriers into the channel (probably blocking them later with the control gate).

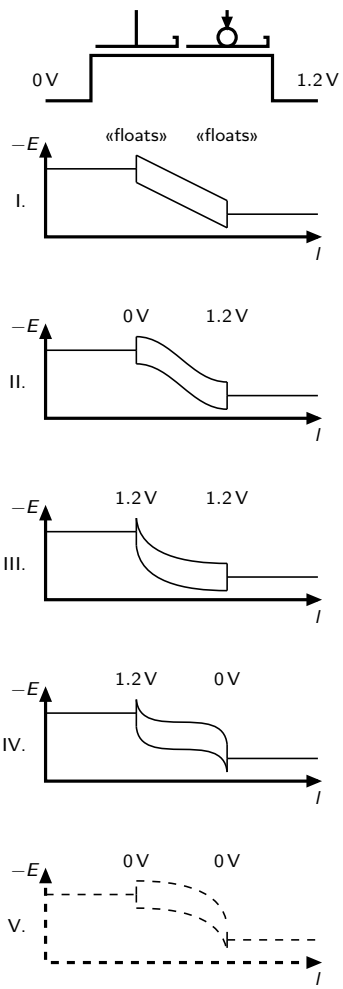


Figure 2.5: Band diagrams

The second band diagram also reveals that band bending is practically symmetric in the closed case. This is the enabler of reconfigurable circuits using this device but will also become a problem for circuit design, as this symmetry lets the device exhibit strong ambipolar characteristics when gate and contact voltages do not align properly (cf. Band Diagram IV). Symmetry of the band gap also dictates, that there is no intrinsic meaning of program gate or control gate to the device. The same way the acting program gate inhibits h^+ charge carriers, the acting control gate inhibits e^- charge carriers. Leaving all voltages the same and reversing the labelling on all terminals would depict a closed PMOS transistor instead of a closed NMOS device.

Band Diagram III shows band bending when the left control gate steers the channel into fully NMOS open by being charged to 1.2 V, as well. The axial length of the band gap at the top left end becomes very narrow, allowing e^- charge carriers to tunnel to the other side and to flow out of the drain contact; h^+ charge carriers are still blocked.

During reconfiguration of a logic gate, the voltage at the program gate might switch faster than supply voltages. So, as a little thought experiment, consider the transistor being in the current NMOS open state. Now, the program gate (right) switches early from 1.2 V to 0.0 V, without the source contact (which is connected to the voltage supply by convention) following suit. This (briefly) leaves the transistor in the state that is shown in Band Diagram IV. The input, which has not changed, still bends the band down while the program gate forces it to bend upwards, thinning out both band gaps and allowing charge carriers from both sides into the channel. Even a supposed inner gate would no longer be able to effectively close the channel at either end, let alone both ends.

This description is concluded by Band Diagram V, which is drawn dashed, and

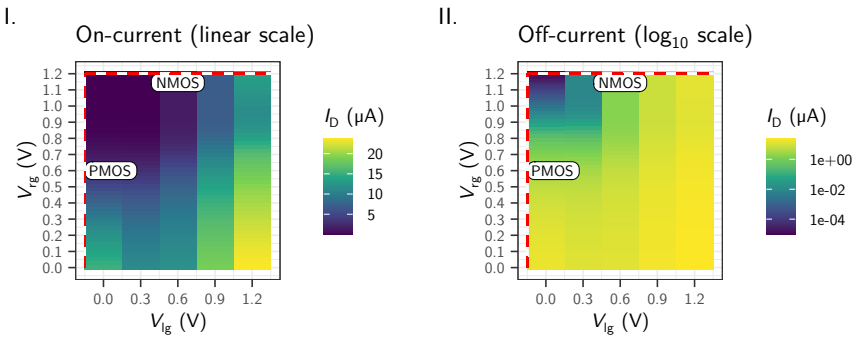


Figure 2.6: TCAD simulation of a single germanium nanowire RFET. 24 nm feature size.

strengthens the view that the device is symmetric. Both transistor gates bend the band upwards. Now, the left gate is blocking e^- charge carriers from entering the channel, which makes it the acting program gate. The right gate opens the channel for h^+ charge carriers, constituting a PMOS open state.

In summary, the device exhibits four extremal states, shown in Band Diagrams II–V. The "off" state and the ambipolar state are symmetric with regard to the functional designation of the transistor terminals. In contrast, the "on" states are distinct states regarding the drawing conventions. Moving the device from one "on" state to another via the "off" state, while avoiding the ambipolar state as much as possible, will be the task of reconfiguration.

Quantitative transistor behaviour Figure 2.6 is based on a technology computer-aided design (TCAD) simulation for a germanium nanowire (GENW) device (similar to the device in [56]), with a channel diameter of 20 nm, a channel length of 48 nm and respective gate lengths of 24 nm. Both the left and the right heat map show the same data, the drain current I_D for varying voltages of the left and right transistor gates. Similar to Figure 2.5, the source/drain voltage is fixed to $V_{DS} = 1.2\,V$. While the left heat map emphasises the on-current using a linear scale, the right heat map allows better judgement of the off-current emphasising small values of I_D with a logarithmic scale. The experiment spans the voltage at the left gate according to $V_{lg} = 0\,V, 0.3\,V, \ldots, 1.2\,V$ and the voltage at the right gate according to $V_{rg} = 0\,V, 0.02\,V, \ldots, 1.2\,V$.

Band Diagrams II–V from Figure 2.5 represent the four corners of each heat map according to the voltages of the left and the right gate. The topmost horizontal line depicts the NMOS behaviour with $V_{rg} = 1.2\,V$ programming the channel and V_{lg} steering the channel open towards $V_{lg} = 1.2\,V$. Likewise, does the leftmost vertical column of values correspond to the PMOS behaviour.

Figure 2.6 I shows that the ambipolar current is significantly stronger than

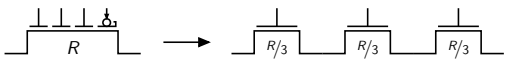


Figure 2.7: Reduced virtual channel resistance compared to a CMOS series of transistors when employing MIGFETs.

either NMOS or PMOS on-current. On the way from either full open state towards ambipolar, the current drops to a minimum at around the threshold voltage (plot corner $-0.3\,\mathrm{V}$) of the current program gate. This is the point where the program gate starts acting as a control gate itself. It no longer blocks unwanted charge carriers and starts injecting them into the channel. This culminates into an ambipolar current that is almost twice as large as either on-current.

Behaviour during reconfiguration Apparently, in Figure 2.6, top left marks the closed transistor state and bottom right the full ambipolar state—under the given source/drain voltage regime, while reversal of V_{DS} would result in a reversal of both corners. Reversing the corners means effectively reconfiguring circuit by switching V_{DS} and V_{rg} in any order. Depending on the current state of the device, e. g. NMOS open (top right), switching V_{rg} first would be a drawback as it would force the device into the bottom right corner. Suppose further that the device part of a CMOS circuit. So, the drain contact is connected to the circuit output and not directly to drain voltage rail, while the source contact is connected to supply. Switching V_{DS} first would move the device state in a fourth dimension of the plot. In Figure 2.6, we assumed $V_{\mathrm{DS}} = 1.2\,\mathrm{V}$. For a source voltage $V_{\mathrm{SS}} = 1.2\,\mathrm{V}$, the drain contact would be at $0\,\mathrm{V}$. So, when changing the voltage at the source contact, its difference towards the drain side drops to $0\,\mathrm{V}$ with no further current flowing.

Multiple independent gates The geometry of nanowire transistors allows them to carry multiple gates. A working Schottky barrier type transistor uses at least two gates, while the FinFET-based devices shown in [65] use at least three. Longer nanowire transistors with four and more gates are conceivable and have been predicted and simulated in [56, 57].

The use of multiple control gates stems from exploiting a feature of Schottky barrier type devices. Their channel current is not so much limited by the intrinsic resistance of the channel, but by the energy barrier itself. Thus, stretching out the channel does not immediately increase the channel resistance to a measurable value. This allows us to put more control channels around the middle of the nanowire between the two Schottky barrier gates without affecting device performance. Additional inner control gates work together with the Schottky barrier control gate in a wired-AND fashion. While one transistor gate can close the channel, all gates have to agree to open it. Looking at it from a circuit designer's perspective, a MIGFET reduces the virtual channel resistance per input as shown

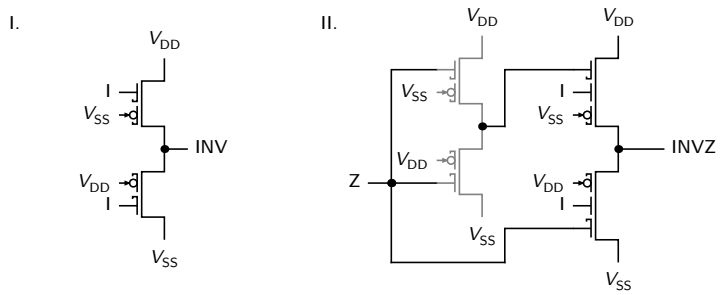


Figure 2.8: I. Smallest possible RFET inverter. II. Inverting tri-state buffer, based on MIGFETs.

in Figure 2.7. Each input has operates a (virtual) transistor with only $1/n \times R$ channel resistance for n inputs. Still, all inputs have to agree to produce an output signal, which brings the total resistance back to R for the on-case. Branches, which are perfectly valid in standard CMOS logic gate design, are obviously not possible—alternatives have to be expressed by fully repeating otherwise shared paths. Multiple independent gates allow a circuit designer to compress unavoidable series paths to a single transistor with the effect of balancing out the output strength for this compressed path against the parallel network on the other side of the complementary circuit design. Thus, they contribute an important function to reconfigurable logic gate design, which is sensitive to these imbalances.

This concludes the description on the basic operation of the transistor device. Its effects on fundamental reconfigurable circuits will be investigated in the following section.

2.3 Fundamental Circuits

Reconfigurable circuits draw their effectiveness from two properties the underlying transistor device should ideally deliver. The first property is the programmable transistor channel, which allows the device to act as PMOS and NMOS transistor by virtue of feeding an extra input signal. The second property, which is delivered by nanowire transistors in particular, is their ability to host multiple control gates without notably impacting device performance. The circuits shown in this section usually make use of both these properties.

Starting out from the simplest CMOS circuit, the inverter in Figure 2.8 I, we can see the added complexity that follows RFET logic gates. In addition to the usual inputs, every transistor has to be connected to both supply voltages to be programmed correctly, one at the source contact and the other at the drain Schottky barrier gate. Before I address reconfigurability by connecting switching input signals to these terminals, instead of static voltages, I will show fundamental logic gates that make use of the multigate property to enhance their functionality.

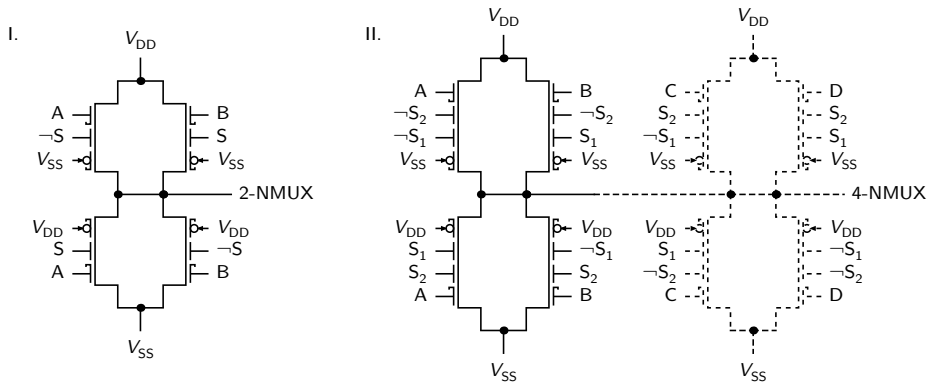


Figure 2.9: The inverting multiplexer. It can be extended by using the selection signals as tri-state control inputs.

Generally, I have chosen to write the name of the circuit's function at the output, rather than a mathematical expression resembling it, at all circuit drawings. The name may be augmented by prefixes declaring inversion (with the letter N), their number of inputs (as n-), and the suffix for tri-state behaviour (letter Z). All of these may be put in parentheses, due to reconfigurability and may reference their reconfigurable input.

The simplest feature-enhanced circuit is the inverting tri-state buffer, INVZ, shown in Figure 2.8 II. It does not use transistor reconfiguration but provides a highly performant implementation of a tri-state buffer due to its use of multiple independent gates. The input signal I is connected to output-side transistors. Apart from the polarity-control gates, the two remaining control gates on each transistor differ in speed and leakage currents. I is connected to the inner gates, as they are the faster ones, while the tri-state signal Z is connected to the outer Schottky barrier gates, which provide low-leakage shut-off. The buffer is designed to provide full output strength by eliminating additional transistors in the driving path – either due to having two transistors in series in each P-/N-network, or a transmission gate at the output. So, this buffer can be used as a drop-in replacement mostly retaining electrical and timing characteristics.

We can extend the idea of the tri-state buffer—deactivating both pull-up and pull-down paths to render a network inert to input voltage changes—into the 2-input inverting multiplexer in Figure 2.8 I. The two selected inputs A and B are distributed to the left and right partial network, respectively, and are controlled the selection signal S and its inverse ¬S. In this particular design, inputs A and B are connected to the Schottky barrier gates to save area; under the assumption of a multiplexer-driven FPGA, like Intel's, inputs A and B are programmed and only input S is time critical. So, this design may be favourable. This design inherits the benefits of the inverting tri-state buffer, its output strength and balance over

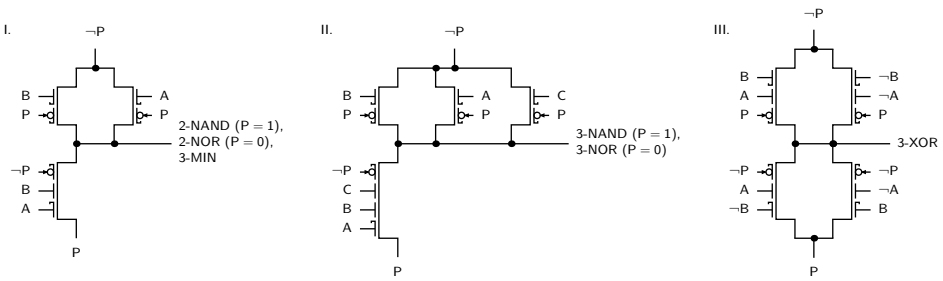


Figure 2.10: Basic reconfigurable logic gates. The NAND/NOR logic gate (I) and the XOR/XNOR logic gate (III) are the smallest reconfigurable circuits. Logic gate II shows that self-dual functions may not have a well-known name, although their parts do.

all signal paths. Additionally, this design can be naturally extended as shown in Figure 2.8 II. By adding another control gate to all transistors, this circuit accommodates a second selection signal S_2, which selectively activates either the dashed or solid set of transistors. Of course, the logic complexity of the selecting between four inputs instead of two must go somewhere else if it is not inscribed in the shape of a multiplexer tree. Hence, it does show up in the quadratic growth of connections to the selection signals S_1 and S_2. So, depending on the application in mind, either design might be beneficial. Nevertheless, cutting the depth of a multiplexer tree in half by providing a breadth factor of four instead of two may outweigh the cost for the selection signals.

Depending on the connections made to this set of transistors, this circuit can perform multiplexer, exclusive OR with two and three inputs, majority, minority and other operations. So, it is a nucleus of reconfigurable circuit design, as it provides just enough flexibility to be usable without impacting circuit performance due to an overly complex design. Additionally, it can have the important property of inverting its input signals. While not as useful in formal logic calculations, this provides many basic logic gates that cover inversion and can be used to cover the full Boolean Algebra—without affecting other Boolean function properties like monotonicity.

The simplest logic gates over two inputs are the NAND/NOR and XOR/XNOR logic gates. In contrast to the multiplexer and the tri-state buffer, they can make use of transistor-level reconfiguration. This is, because their functions each correspond to one half of larger self-dual Boolean functions, 3-MIN and 3-input exclusive OR (3-XOR) respectively. Using reconfiguration, they can be implemented as the circuits shown in Figure 2.10 I and III. The NAND/NOR logic gate in its form given in Figure 2.10 I was first introduced in [23], where it still used exclusively four 2-gate RFETs, and was brought to its shown form in [57]. This logic gate uses a single input, P, to drive the reconfiguration. When this signal is understood as a "program" signal, it programs the circuit to either NAND or NOR function, similar to programming an FPGA or CPLD. Due to the nature of transistor-level

reconfiguration, though, the input signal can also always be understood as an additional input. So, this completes the NAND/NOR logic gate to a 3-MIN logic gate.

This construction of a physical circuit template gets functionalised by attaching the inputs not only to the normal control gates but also the program gates and source contacts. From this circuit, we can get a first intuition of what transistor-level reconfiguration is able to achieve in larger circuit designs. It is less of a radical new circuit architecture, like the idea of using lookup tables to express Boolean functions and well-structured pass-gate logic to connect them. In contrast, it is more of a functional compression approach that allows a physical circuit layout to achieve more with the same structure by using its connections in a flexible way.

The circuit depicted in Figure 2.10 II shows a natural extension to the 3-MIN logic gate. It represents a reconfigurable 3-input NAND/NOR logic gate but otherwise has no well-known name on its own. Adding a third transistor in the parallel network would be achievable in standard CMOS as well. The real benefit—next to the logic gate being reconfigurable—lies in its series path, which can be extended without impacting circuit performance. Adding a second logic stage and replicating shared structure to the other half of the decision tree can be deferred for another input, which keeps the overall circuit smaller and more efficient. This circuit comes in 20 architectural variants.

Figure 2.10 III picks up the arrangement from the inverting multiplexer in Figure 2.8 II, and is merely routing the inputs to different terminals implements the new function. It was first shown in [57, 46] and is a variant of an earlier design shown in [65] and Figure 2.11 I. Both designs are competitive, each having their benefits and drawbacks depending on the performance measure and circumstance, and both were conceived on paper by method of taking a sharp look. As I will show in detail in Chapter 5, there are various other, surprising variants with also beneficial performance properties. In contrast to the former two variants, these are systematically produced and analysed, resulting a clear picture of the design space.

In this section, I want to pick out three variants of the 3-XOR logic gate to show another structural design decision that influences the performance of reconfigurable circuits.

The designs I and III shown in Figure 2.11 are both competitive, while the design II turned out to be worse in every aspect. The numbers below the circuits show the worst-case delay, worst-case power dissipation, and worst-case energy consumption per operation. While the first number of each pair shows the measure for input A being fixed, the second number shows the measure for the worst combination of inputs switching. The first take from this is, that multi-stage 3-XOR designs are not beneficial, when MIGFETs are available, opposite to standard CMOS technology.

The first circuit variant is the aforementioned variant from [65] by EPFL and connects input A solely to the source contacts, but not the transistor program gates. This is possible due to the duplication of input B at both Schottky barrier

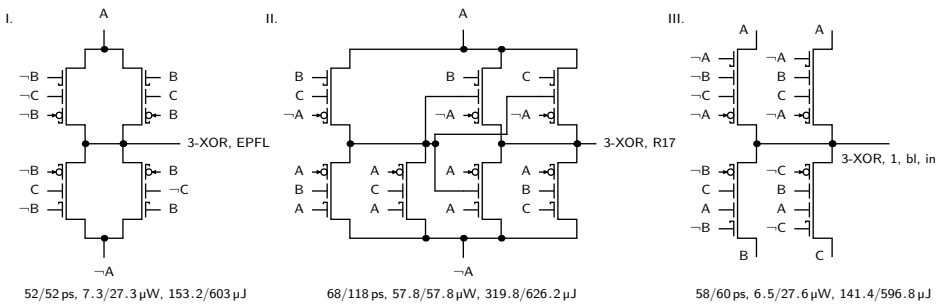


Figure 2.11: 3-XOR logic gate variants. I. was shown in [65]. II. was shown in [46] as a 2-stage variant, that avoids negating inputs B and C. III. was found via design space exploration shown in Chapter 5. Numbers at the bottom show worst-case results for delay, power dissipation and energy consumption per operation.

gates at all transistors to avoid the ambipolar state. The design is very competitive, scoring first in worst-case delay, and in the Top Ten for power dissipation and energy consumption. But the important fact for this section is, that it demonstrates one reconfiguration mode that I will call *transmission gate mode* for the remainder of this work.

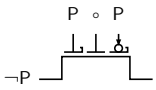
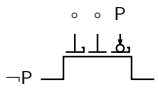
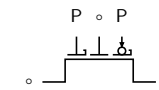
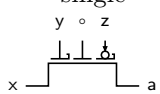
The second mode is shown in the 2-stage variant, which does not duplicate the signal at the program gate to the other Schottky barrier gate, when there is a third signal that can take this spot. This scheme generally results in very low power dissipation within the same circuit variant. I will call this connection mode *single gate mode*.

The third mode always duplicates the input to the program gate at the other Schottky barrier gate. This forces all other inputs to the inner gates, which is why I call this the *inner gate mode*. The quadriga of reconfiguration modes is completed with the illegal variant, which I will call *ambipolar mode*, in which the there is no relation between the source contact and the Schottky barrier gates. As this may result in ambipolar transistor behaviour, this mode is usually useless for circuit design. The terminology was first introduced by Steffen Märcker in [34].

All three modes have their advantages, depending on the environment that the circuit is to be used in and their usefulness, especially of the transmission gate mode, is highly dependent on the Boolean function of the circuit. While it performs excellent for the 3-XOR, it is dysfunctional for the 3-MIN logic gate that I show in [47].

The reason is that in transmission gate mode, the reconfiguration input P is independent of the source signal and the drain signal. So, although, by connecting signal P to both Schottky barrier gates, the transistor does not enter ambipolar states, it will reverse direction, because the source contact signal no longer limits its conducting states. In this configuration, the transistor may either conduct from left to right, when all signals at the transistor gates agree and are opposite of the

Table 2.3: Transistor reconfiguration modes.

Source contact	Polarity control signal P	
	Both SB gates	Single SB gate
$\neg$P	 inner	 single
Other ∘	 transmission	 ambipolar

signal at the left contact. Likewise, due to the same argument, it may also conduct from right to left regardless of the state of the left signal. In complementary circuits, this could lead to unwanted short circuit currents, depending on the Boolean function it implements. A Boolean function that can be implemented using the transmission gate connection mode must have the following property: Whenever all signals $\mathsf{S}_1, \ldots, \mathsf{S}_k$ connected to the transistor gates are at the same value, potentially opening the transistor channel, the *intended* output value (at the drain contact) must correspond to the *current* value at the source contact S_s. The following proposition must hold for the Boolean function:

$$\forall i \in \{1, \ldots, k\}, s \neq i.\ \mathsf{S}_s \oplus \mathsf{S}_i$$

In this chapter, I demonstrated the kinds of reconfigurable circuits that can be constructed from polarity-controllable transistors. After explaining the electrical function of the device, I determined its effect on reconfiguration and the conditions that are necessary to facilitate efficient reconfigurable circuit designs, by showing examples of fundamental designs. I will defer the detailed design space exploration and performance evaluation of single logic gates to Chapter 5 after the tooling, the exact measures and methods to evaluate these measures were properly introduced in Chapter 4. The next chapter establishes transistor-level reconfiguration in the larger scheme of combinational circuits.

Chapter 3

Combinational Circuits and Higher-Order Functions

The fundamental circuits presented in the previous chapter show interesting characteristics. They are especially compact and use fewer numbers of transistors than standard CMOS implementations. The next questions, I want to investigate, are, how well they would perform in larger explicitly reconfigurable designs, and what their influence is on the characteristics of larger combinational circuits. This investigation, described in the following sections, was done very early in the development of the silicon- and germanium-based transistor devices. Thus, no reliable electrical data was available at the time, and the device was nowhere near a top-down production-ready state, which could have rendered it compatible with established CMOS devices at the time. This is why I chose to evaluate circuit performance using Logical Effort Theory introduced by Ivan Sutherland in [53]. It enables me to capture relative performance characteristics and to show the development of circuit performance from small to larger circuit designs without the need for electrical device data. The following sections capture the design and evaluation of programmable logic cells suitable for augmentation of ASIC designs, their basic design and improvements with RFETs. They are followed by an investigation of a combinational adder design based on the conditional sum adder.

3.1 Programmable Logic Cells

The previous chapter demonstrated fundamental reconfigurable circuits eligible to be collected in a process design kit (PDK). To further discriminate the meaning of reconfiguration by its intended use, we can say that the fundamental circuits shown earlier employed *implicit reconfiguration*. Those circuits were used according to their function and reconfiguration was merely a vehicle to achieve that goal. This section concentrates on multi-functional circuits, whose purpose is to

provide a set of Boolean functions, which are known to facilitate the implementation of larger designs. These circuits set themselves apart from the fundamental circuits by providing explicit selection inputs that allow a designer to choose from the set of functions, comparable to macrocells and configurable logic blocks (CLBs). The selection inputs may reconfigure the multi-functional circuit. I will, therefore, refer to this type of reconfiguration as *explicit reconfiguration*. The following section highlights the benefits of implicit reconfiguration by compressing functionality into smaller logic gates.

3.1.1 Critical Path Delay Estimation using Logical Effort Method

In the early stages of technology development – a stage in which this thesis is written concerning nanowire-based reconfigurable transistor technology – simulation program with integrated circuit emphasis (SPICE) network analysis is not feasible, because it is based on the availability of a SPICE transistor model. Simpler methods are Elmore delay [12] or algorithms using the more general Padé approximation [43, 40]. Both methods perform *moment matching*, a method for asymptotic waveform evaluation. The real transfer function (that describes the delay) is approximated by a rational function $H(s)$ with the restriction, that the real function and the approximation agree at $x = 0$ and at their derivatives at $x = 0$. The derivatives of $H(s)$ at $s = 0$ are the coefficients of the Maclaurin series of $H(s)$ and are called *moments* of the transfer function, hence the name moment matching. Elmore delay only matches the first moment m_0 while Padé approximation matches as many moments as necessary to gain a required accuracy. See also [14] for further details on the method. Both methods require some kind of PDK that provides the logic gate delays and/or transistor models like in SPICE analysis. The Logical Effort Method [53] provides a simple way to calculate gate delays and is also suitable for circuit delay calculations for small circuits that are not dominated by wire delays. I used this method for both early estimations of logic gate delay and for calculating the critical path delays of the circuits presented in this chapter.

Logical effort calculation for multistage logic circuits with N stages is represented by the normalised path delay D driving a fanout H:

$$D = N \times \sqrt[N]{F} + P \tag{3.1}$$

$$P = \sum_{i}^{pcrit} p_i \tag{3.2}$$

The normalised delay D is the geometric mean of N individual stage efforts along the critical path *pcrit* (an ordered set of logic gates), culminating into the path effort F. It also includes the intrinsic parasitic delay P which is the sum of the parasitics p_i for each of the logic stages. Path effort F is computed from the path electrical effort H (also called fanout), the path logical effort G and the

branching effort B:

$$F = G \times H \times B \tag{3·3}$$

$$G = \prod_{i}^{pcrit} \frac{C_i^{\text{in}}}{C_{\text{INV}}^{\text{in}}} \tag{3·4}$$

$$B = \prod_{i}^{pcrit} \left(1 + \frac{C_i^{\text{rest}}}{C_i^{\text{path}}}\right) \tag{3·5}$$

The logical path effort G describes the structural effort along the path by the relation of the input capacitance C_i^{in} at a particular stage i to the inverter input capacitance $C_{\text{INV}}^{\text{in}}$. So, when a signal needs to be fed to more transistor gates, the whole logic gate at that stage is expected to be structurally slower. By restricting to the critical path, which is confined to a single input, the method neglects dynamic effects that arise from multiple active electrical paths by multiple inputs switching simultaneously. Likewise for the branching effort B, logical effort method determines how much effort in each stage needs to be devoted to drive logic gates *other* than the next one on the critical path, i. e. how much *more* load is put on the critical signal. Finally, the approximated absolute delay t is calculated from the normalised delay and the intrinsic delay τ:

$$t = \tau \times D \tag{3·6}$$

$$\tau \approx \frac{V_{\text{DD}}}{I_{\text{D}}} \times C_{\text{gate}} \tag{3·7}$$

τ expresses the input / output delay of a standard-sized inverter of a particular technology. It is conceptually described as delay experienced by the inverter driving an identical second inverter. The delay is taken starting from the input transient crossing $V_{\text{DD}}/2$ to the output transient also crossing that point. These delays are usually hard to obtain for new technology, because no reliable circuit simulations exist, yet, and it is the whole point of Logical effort theory to have very little reliance on simulation. This is reflected in the computation of Equation 3·7, which is an approximative solution for the actual inverter delay. V_{DD} is a technology parameter and I_{D} can be found via device simulation in a technology simulator like TCAD. C_{gate} is hard to retrieve and is usually indirectly inferred from device simulation.

The benefit of the logical effort method over the other delay approximation approaches is, that its results are technology agnostic. This was important in the course of this work because most technology parameters of nanowire transistor technologies were either preliminary, unreliable in themselves or outright non-existent. Especially C_{gate} is hard to obtain in early technology evaluation and so is the inverter delay. Thus, I have chosen to work with the shown approximation for τ and to mostly work with the normalised delay D.

For a single-stage logic gate, the formula for the normalised delay simplifies to d, the path effort F simplifies to the stage effort f and the parasitics p are directly

taken for the single stage. Due to $N = 1$, the root becomes trivial and *pcrit* is the set containing only said logic gate. Branching effort $B = 1$, because is no C^{rest} which lies off the path. The single stage normalised delay is, thus, computed by:

$$d = f + p \qquad \text{with:} \quad f = g \times H \tag{3.8}$$

Table 3.1: Comparison of static logic gate implementations in MIGFET and CMOS technology shown in [57]. Absolute delays determined by model checking explained in Chapter 4.

Gate	Transistor cnt. # T Logical effort g Parasitic delay p Normalized delay d	MIGFET GENW (24 nm) Delay t (ps)	CMOS FDSOI (32 nm) Delay t (ps)
INV	# T	2	2
	g total	1	1
	p	1	1
	$d^{H=1}$	2 — 24	2 — 9
2-NAND	# T	3	4
	g total	2	2 2/3
	g per input	1	1 1/3
	p	1 1/2	2
	$d^{H=1}$	2 1/2 — 32	3 1/3 — 13
2-NOR	# T	3	4
	g total	2	3 1/3
	g per input	1	1 2/3
	p	1 1/2	2
	$d^{H=1}$	2 1/2 — 32	3 2/3 — 14
2-X(N)OR	# T	4	8
	g total	4	8
	g per input	2	4
	p	2	4
	$d^{H=1}$	4 — 44	8 — 22
3-MIN/ 3-MAJ	# T	6	10
	g total	6	12
	g per input	2	4
	p	3	6
	$d^{H=1}$	5 — 36	10 — 23
2-NMUX	# T	6	12
	g total	4	8
	g per input	1	2
	p	2	4
	$d^{H=1}$	3 — 38	6 — 19

Table 3.1 shows a comparison for a selection of the fundamental logic gates that were used in implementing the multi-functional circuits of the following section. The table lists the implementations as depicted, using MIGFET transistors, and compares them with functionally equivalent implementations using standard CMOS transistors. Jens Trommer and myself feature this table in [57] to show a first approximation of the capabilities of multigate transistor circuits in general and reconfigurable circuits, in particular. The gates must only be cautiously compared horizontally because MIGFETs and CMOS devices have very different intrinsic delays. Thus, I have added a second column for each transistor technology, showing the actual delay, as determined by the approach further developed in this thesis (see also Chapter 4). Nevertheless, the table makes a structural comparison feasible that highlights the benefits of multiple independent gates per transistor, reducing the parasitic delay p, as well as the benefits of reconfigurable circuits (starting from 2-X(N)OR downwards), having a similar profound effect on the logical effort g and the number of transistors # T.

As Table 3.1 shows, both inverters are structurally equivalent. To get the best comparison between two very different technologies, I use the 24 nm GENW described in the previous chapter, and a 32 nm CMOS technology described in [41] that is known to be used in scaled 24 nm processes in production. Both technologies are modelled in my modelling language described in Chapter 4, as are the circuits. The actual delay shown next to the normalized delay is computed from those models. The difference in absolute delay is the measure for the intrinsic delays τ of each technology with a relation of $2.7 : 1$. For each circuit, the table lists the number of transistors (as a rough estimate of the area), the total logical effort (g total) and logical effort of the worst input (g per input) ignoring the reconfiguration signals. This means, that for CMOS to implement the same reconfigurable circuit like 2-X(N)OR etc., it may need many more transistors than the RFET implementation. Still, both implementations feature the extra reconfiguration input that allows a designer to select the function of interest. In case of CMOS it merely turns off one set of transistors and enables another, while in case of RFETs, it actually reconfigures the transistors. The table also shows the parasitic delay and, as mentioned earlier, the normalized delay for fanout $H = 1$.

Looking at the normalized delays and absolute delays of the inverter, it becomes apparent how large the difference can be. So, how good does logical effort predict the behaviour of RFET technology circuits?

So, for the RFET implementations the intrinsic delay has the magnitude $[\tau_{\text{RFET}}] = {}^{24}/_{2} = 12$, whereas for CMOS it is $[\tau_{\text{FET}}] = {}^{9}/_{2}$. The delay factor k between the two technologies is:

$$k = \frac{t_{\text{INV,RFET}}}{d_{\text{INV,RFET}}} \times \frac{d_{\text{INV,FET}}}{t_{\text{INV,FET}}} = \frac{24\,\text{ps}}{2} \times \frac{2}{9\,\text{ps}} \approx 2.7 \tag{3.9}$$

By rule of proportion, the prediction for the absolute delay of the RFET implementation of the 3-MIN circuit is the product of delay factor k corrected by

the real intrinsic delay for the 3-MIN circuit:

$$t_{3-\text{MIN,RFET}} = k \times d_{3-\text{MIN,RFET}} \times \frac{t_{3-\text{MIN,FET}}}{d_{\text{MIN,FET}}} \tag{3.10}$$

$$\approx 2.7 \times 5 \times \frac{23\,\text{ps}}{10} \approx 31\,\text{ps} \tag{3.11}$$

In want of a better estimate, the correction for the real intrinsic delay was taken from the field-effect transistor (FET) implementation. Though, looking at the absolute delay for the RFET implementation of 3-MIN with 36 ps, the prediction is quite true to the actual delay, being 16 % off. The absolute delay was determined by model checking which is explained in Chapter 4, further values for various other logic gates are shown in Table 3.1. This table serves as the motivation for showing the principle effectiveness of reconfigurable standard cells and the complexity of comparing actual delay results that stem from the different transistor technologies. Neither is comparing the delays for a single output load nor abstracting from absolute delays to structural delay enough to give a clear picture. Additionally, as it turns out in Chapter 5, there are usually numerous reconfigurable implementations for a single logic gate, sometimes with very different delay characteristics and sensitivities to changing output load.

3.1.2 Multi-Functional Circuits

Multi-functional circuits form the basis of FPGA circuits. They can be implemented based on multiplexer trees or memory decoder logic, and, of course, as complete crossbar networks as shown in Figure 2.1. Each approach has its own drawbacks and benefits with crossbars being the hardest to translate into an ASIC design pattern. This section investigates designs based on multiplexer trees that provide simple reconfigurable cells for use in ASICs.

Jens Trommer and myself have shown in [57] how to improve the 6-functional circuit depicted in Figure 3.1 α. The Figure shows three implementation variants α, β and γ. They are functionally equivalent, but for the sake of comparability, implementation α, which is standard CMOS, also uses inverting multiplexers. Implementations β and γ are targeted for RFET and MIGFET devices and they use the logic gates from Table 3.1. Although the use of inverting multiplexers comes as a drawback for CMOS implementation α, this still makes sense given the fact that transmission gate multiplexers would force the outputs of the gates in the left column through two, respectively three, transmission gates, diminishing signal quality. Nevertheless, the true sweet spot for a CMOS implementation will be in a mix of multiplexer (MUX) implementations. All networks have two inputs A and B and select the function via the three selectors S_1–S_3.

The CMOS implementation α uses 92 transistors compared to 34 transistors used by implementation β, which uses the logic gates shown in Table 3.1. The two outer branches of implementation α, featuring the NAND and NOR logic gates,

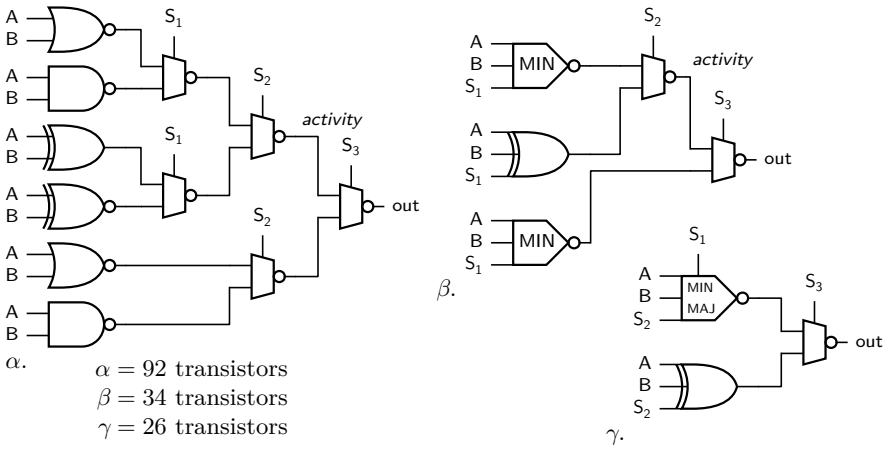


Figure 3.1: 6-functional programmable logic circuit controlled by a MUX tree. Implementation α is standard CMOS, while Implementations β and γ use reconfigurable logic gates from Table 3.1.

where placed at different depths of the multiplexer tree, such that one branch effectively computes the AND and OR functions, while the other computes the NAND and NOR functions. It is evident, that the two top-left NAND and NOR logic gates and the multiplexer can be merged into a single 3-MIN circuit. The same holds for the two lower left logic gates. In the same way, the branch containing the 2-XOR and 2-XNOR circuits can be merged, as well. The inverter-buffered transmission gate multiplexers used in implementation α can be replaced with the efficient static variant shown in Figure 2.9 now, that MIGFETs can be used for its implementation. This saves 63 % transistors and, thus, static power dissipation, but it is to be noted that MIGFETs are larger than CMOS devices. So, area reductions are less. It also reduces the circuit from four to three logic stages, moving selector S_1 nearer to the critical path. Having S_1 as an input increases the balance of the overall circuit and the performance of inputs A and B, in particular, but it sacrifices speed for switching scenarios in which A and B are quiet. In implementation α merely the (speculatively) computed result needs to be selected, while in implementation β the function must be recomputed altogether. Nevertheless, under the assumption of explicit reconfiguration, the reconfiguration signals are not regarded as timing critical, because they are not part of the user logic but a static part of an already configured device.

The second optimization is shown in Figure 3.1 implementation γ. The AND, OR, NAND and NOR functionality can be completely subsumed by utilizing a 3-input MIN/MAJ logic gate. This logic gate realises inversion and selection of AND and OR functionality in a single stage. It further reduces the circuit size to 26 transistors, while reducing the number of logic stages to two. Similar to imple-

mentation β, selector S_2 is also elevated to an input – selector S_1, although shown at the top of the 3-input minority/majority reconfigurable logic gate, works at the same logic stage as the left inputs. This implementation is further reducing the number of speculatively active logic paths, reducing the overall dynamic power dissipation.

TABLE 3.2: COMPARISON OF THE PATH DELAYS D FOR THE MULTI-FUNCTIONAL CIRCUITS SHOWN IN FIGURE 3.1. BOLD RESULTS SHOW THE CRITICAL PATH DELAY, ITALICS DISPLAY CRITICAL PATH INVOLVING A SELECTION INPUT.

Signal	Impl. α Path delay D $H = 1$	Impl. α Path delay D $H = 4$	Impl. β Path delay D $H = 4$	Impl. γ Path delay D $H = 4$
$\mathsf{A/B}_{\text{NAND/NOR}}$	21.5 / 22.1	26.1 / **27.0**	**21.5**	18.5
$\mathsf{A/B}_{\text{AND/OR}}$	19.7 / 20.5	25.5 / 26.7	21.1	**21.1**
$\mathsf{A/B}_{\text{XOR/XNOR}}$	**22.7**	**27.0**	20.3	19.0
S_1	16.8	21.7	20.1	18.5
S_2	15.2	22.0	20.7	*22.7*
S_3	15.4	*28.5*	14.3	14.3

Using logical effort theory, we can now calculate the path efforts of the individual input-to-output paths for all signals. Table 3.2 shows two sets of results for implementation α for paths involving (N)AND/(N)OR logic gates, due to their deviating efforts. Also, implementation α is shown with a fanout $H = 1$ and 4 while the reconfigurable implementations are just shown for fanout $H = 4$. This is to show, how the critical path may shift from a user input to a reconfiguration signal albeit the signal not being involved in computation but only selecting speculative results. The bold numbers represent the critical path delays D as normalised delays while the italic numbers represent the worst case path delays when they involve a selection input. Under the assumption of explicit reconfiguration, though, these numbers can be ignored in favour of the smaller bold results.

As shown earlier, although GENW devices may be the slower technology to begin with, their logic gate implementations show strong benefits regarding the logical effort that is needed to perform their function. With gains of up to 25 % compared to a CMOS implementation, this holds true for more complex circuits like the multi-functional circuits, as well. Additionally, for implementations β and γ, the resulting path delays are very close to each other, which is a highly desirable property for fast circuit design. The development of the path delays for implementation β and γ show that, considering all path delays, implementation β seems to be the sweet spot where all delays are closest together and smaller than $D_{\gamma}^{H=4}(\mathsf{S}_2) = 22.7$. According to logical effort theory, this indicates that the CMOS circuit design was too deep for its complexity. We fixed that by widening it by compressing the function, exploiting transistor reconfiguration and multiple gates per transistor. Slightly over-optimizing implementation γ, it is certainly the

smallest implementation and has also the edge in delay when considering explicit reconfiguration.

Circuit Energy Consumption Estimation Following the observations how logical effort develops when changing the implementation paradigm to reconfigurable circuits, which focused on making delay predictions structurally comprehensible, it would also be interesting to see how circuit power dissipation and energy consumption develops for reconfigurable designs. Dynamic energy consumption can be judged with the dynamic power product formula, which delivers Joules per Operation:

$$\sum C \times V_{\mathrm{DD}}^2 = \left(\sum C_{\mathrm{dyn}} + \sum C_{\mathrm{static}} + \sum C_{\mathrm{int}}\right) \times V_{\mathrm{DD}}^2 \qquad (3.12)$$

Capacitance C is the input capacitance of the NMOS transistor. It can be split into three sets of capacitances, the dynamic and static gate capacitances C_{dyn} and C_{static}, and the inter-nodal capacitances C_{int}. They are hard to obtain and were inferred from diagrams in [41] for the CMOS implementations and taken from similar TCAD simulations for the GENW devices that were used to describe the device characteristics in the previous chapter. The formula is pessimistic in the sense that all possible transistor gates are considered in the calculation. Though, it is known, that not all gates will participate in any single switching operation. For the multi-functional circuit, the results of this computation are shown in Table 3.3.

Table 3.3: Dynamic energy consumption using normalised capacitances that allow a structural comparison between technologies.

	Impl. α	Impl. β	Impl. γ
C_{dyn}	92	54	46
C_{static}	0	24	16
C_{int}	22	6	4
V_{DD} (V)	1	1.2	1.2
$\sum$	114	121	95

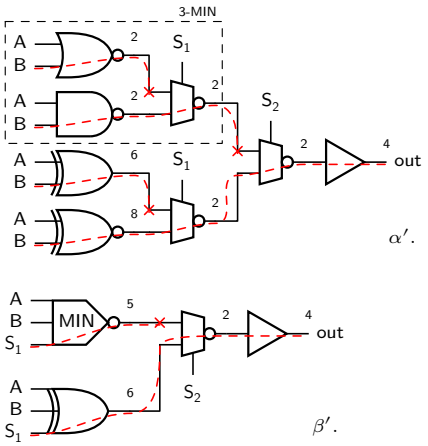
The MIGFET-based implementations have no inter-nodal capacitances, because all paths with two or more transistors in series could be eliminated. Unlike implementation α, though, they need additional transistor gates that driving polarity control (i.e. circuit configuration). Implementation β is on par with the CMOS implementation with only 5 % deviation. Nevertheless, this estimation is quite rough and neglects the possibility of prolonged cross currents due to shallower transients during circuit switching. We have seen earlier in Table 3.1 that the MIGFET device start with a quite larger intrinsic delay compared to CMOS. Later analysis with the model checking-based approach, which I will lay out in the next chapter, will show that especially the static logic gate implementations for 3-MIN

and 3-XOR of this section suffer from high energy consumption although, they can trade this in for a very low worst-case delay. Implementation γ can excel in both normalized delay and energy consumption due to its compact design and shallow tree of logic gates.

The dynamic power formula is not the only way to analyse the expected energy profile of a circuit. In my joint work with Shubham Rai et al. [45], I showed that transistor activity can be used as a measure that models worst-case dynamic energy consumption. Activity describes the number of transistors of a logic circuit that are excited to switch by a single input change. The worst-case activity is maximum number of transistors that can be excited by a single input change and depends on the input pattern before the change and the logic function (for all non-trivial functions).

In Figure 3.1, implementations α and β show a label *activity* at the output of a particular multiplexer. Both circuits perform the same function up to this point, but due to their different implementations, need different numbers of transistors to perform it. Figure 3.2 depicts this function as implementation β'. In contrast to the sub-circuit shown in β, this implementation does not use the static logic gate implementations from this chapter but size optimized variants for the 3-MIN and 3-XOR logic gates, which are only available to multiple independent gate RFETs. The dashed red lines show those paths through the tree that excite the worst-case activity in the logic gates. Selector S_2 selects the output of the 3-XOR logic gate to recall the critical path, which leads from Selector S_1 to the output. Nevertheless, the same selector also (accidentally) excites the maximum activity in the 3-MIN logic gate. (Although this could be remedied by reordering input S_1 at one of the logic gates, this would negatively impact worst-case delay of either input A or B, making it a choice that depends on the intended application.) To give a realistic comparison between this implementation and the sub-circuit of implementation α (called α'), both circuits drive the output through a buffer, adding another four active transistors. This (energetically) simulates a load of two inverters.

As the table in Figure 3.2 shows, this MIGFET implementation uses only 61 % of the transistors in the worst-case, compared to CMOS, to perform the same function. This is quite a gain compared to the pessimistic calculations using the dynamic power product formula. The last line of the table shows the prediction delivered by the modelling approach discussed in the next chapter. It is not capable of directly modelling a circuit of the size shown in Figure 3.2 but is suitable for single logic gates, which is why the results for the 3-MIN gate are shown here. Like the columns in Table 3.1 that show the absolute delay to put it into perspective to the normalised delay – arguably the MIGFET circuits lost quite strongly against CMOS, there. The last line shows just how much more power efficient the GENW transistor device is to a high-speed off-the-shelf CMOS device. The technology can perform the same operation using only 1/3 of the energy; and this is not the most energy-efficient implementation of that logic gate, while there are only minor optimizations left for the CMOS implementation. So, regardless of



α'. β'.

	Impl. α'	Impl. β'
Transistor count # T	62	25
Normalized delay D	28.3	19.13
w/c Activity	28	17
$E^{H=1}_{\text{3-MIN}}$ (pJ)	960	298

Figure 3.2: Worst-case activity in the (partial) multi-functional circuit. Red dashed lines show the active paths through the circuit contributing to worst-case activity. The numbers show the numbers of active transistors. Last line of the table shows the energy consumption of the 3-MIN sub-circuit, determined by `prism-gen`, the circuit modelling approach described in Chapter 4.

the intended application, MIGFETs provide viable alternative implementations of useful logic circuits that, in addition to the already known low-power design of the transistor device itself, feature significant energy savings over CMOS circuits.

3.2 Improved Conditional Carry Adder

Implicit reconfiguration is another interesting use-case for reconfigurable transistor devices. It is obvious now, that there is no single useful concept of reconfiguration that could refute rivalling definitions as mere computation. Of course, the inner workings of the reconfigurable circuits shown up to this point can all be explained with the same concepts that apply to "non-reconfigurable" logic gates. The reason, though, is just, that on the level of transistor reconfiguration, both concepts, computation and reconfiguration, can be commonly supplanted by the single concept of Shannon expansion. Then again, as long as we are not considering reconfigurable signal routing, all logic circuits can be described in terms of Shannon expansion, be it configurable logic blocks in FPGAs or macrocells in CGRAs or the multi-functional circuits shown in this chapter. Implicit reconfiguration, then, should be thought of as a means to compress a logic function into a smaller set of transistors than would be possible by standard CMOS by exploiting transistor-level reconfiguration.

In my work with Jens Trommer et al. [46], I investigated the possibilities to optimize a larger arithmetic circuit. I chose a variant of the conditional sum adder,

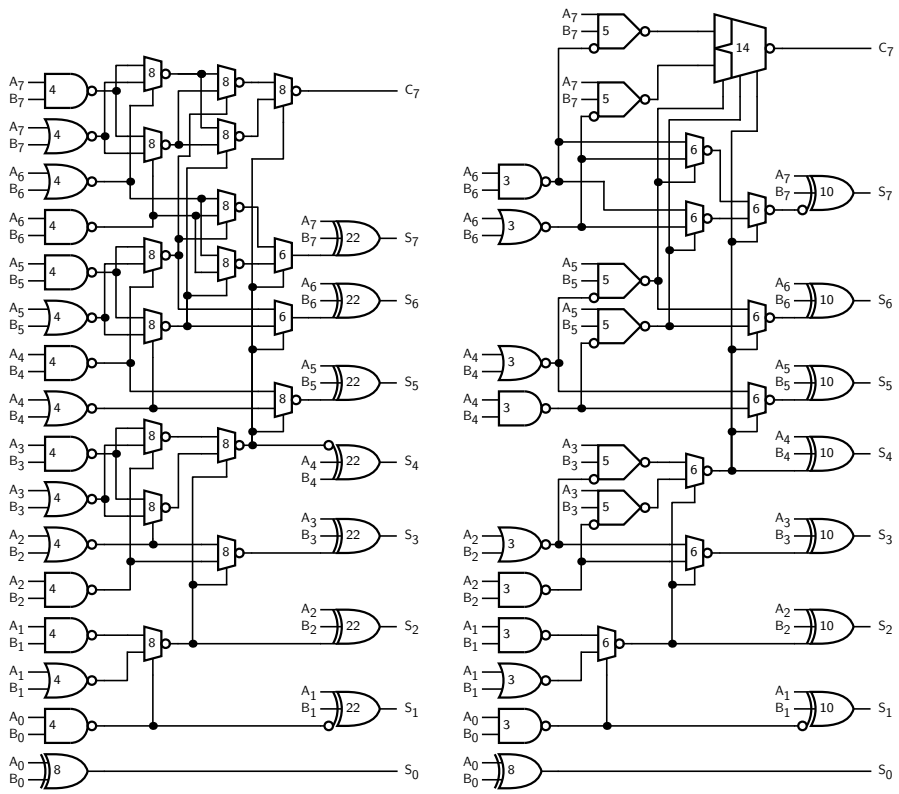


Figure 3.3: 8-bit conditional carry adder. *Left* the CMOS implementation. *Right* the MIGFET implementation with several replaced logic gates. Numbers inside show the number of transistors per logic gate.

which was already optimized by stripping the multiplexer network, which is used to communicate the conditional sums, to the bare minimum to only communicate the conditional carry signals. It was shown in [9] and was named conditional carry adder (CCA) by its authors. The left circuit in Figure 3.3 shows a depiction of an 8-bit CCA. The circuit lent itself because of its regular structure featuring several NAND/NOR/MUX blocks at the input side that we have already investigated in this chapter. It also shows a distinct pattern in the multiplexer network that can be first seen in the calculation of the output carry signal C_7. The numbers inside each logic gate represent the number of transistors that are needed to implement it.

To support the critical signals (the ones with the least amount of slack in relation to the critical path), which are the carry propagation signals, the CMOS implementation also uses inverting multiplexers for the most part. Arithmetically, an inverted multiplexer output in stage k which feeds a selector in stage $k+1$ can

be addressed by exchanging the two signals that are selected in stage $k+1$. This means (apart from the extra two transistors), using inverting multiplexers inflicts no additional cost for the circuit implementation. For the CMOS implementation, there is one exception, though, at the top input to the 3-XOR logic gate producing S_4. In spite of similar occasions in the MIGFET implementation on the right, the input inversion to 3-XOR gate is a real inverter – whereas for the MIGFET implementation, the shown inverted inputs could all be achieved by rewiring the signal inside the logic gates without adding another inverter.

Figure 3.3 *right* shows the MIGFET implementation, and it is apparent that only some of the triplets could be exchanged for significantly smaller 3-MIN logic gates. The shown implementation tries not to make sacrifices regarding critical path delay but one. Leaving out the specialised multiplexer gate producing C_7 in favour of the usual multiplexer tree would give a slight improvement, but I kept it in this design, because the pattern causing it starts to appear more often for larger circuits, like the 32-bit variant shown by its original authors. Also, it shows an interesting optimisation applicable to this circuit. All of the 3-MIN circuits were placed away from the critical path, which is from input $\mathsf{A}_0/\mathsf{B}_0$ to output S_7, because they slightly increase the delay for the carry propagation signal. The implementation also avoids 3-MIN circuits in places like the inputs $\mathsf{A}_2/\mathsf{B}_2$, although they form triples with the multiplexers in the third column (counted from the input). As can be seen, they not only feed into the multiplexers but are also into the 3-MIN gates of inputs $\mathsf{A}_3/\mathsf{B}_3$. Table 3.1 showed that the logical effort for the 3-MIN gate is higher than for the simpler 2-NAND/2-NOR gates. So, the chaining of two 3-MIN gates had to be avoided in those cases. For larger circuits, though, the higher bits of the adder develop enough slack such that chaining of multiple 3-MIN gates becomes feasible for additional savings and a more balanced circuit.

The logic gates in the MIGFET implementation are the ones shown in Figure 2.9 I for the 2-inverting multiplexer (NMUX) gate, Figure 2.10 I for the 2-NAND/NOR and 3-MIN gates and circuit III for the 2-/3-XOR gates. One notable exception is the top-right logic gate producing C_7. It implements a pattern that also appears more often for larger input sizes, which is related to the underlying information density of the addition operation.

Logic circuit implementation cannot escape the fundamental limits of information density, which itself is the driving force of signal communication. Every complete and correct implementation of an algorithm must communicate as many bits of information to as many sub-circuits as are necessary to fulfil its definition. Most implementations communicate more bits, 1) due to a non-trivial mapping between electrical signals and bits of information and 2) due to restrictions of the underlying implementation vehicles, the transistors. As this work already showed, reconfigurable transistors allow for a reproduction of a tighter mapping between information bits and electrical signals. The same is true for adder circuits. Not all bits of inputs can influence all bits of output in the same way, otherwise the critical path would have to grow linearly in the number of inputs, which it does not. The best known adder designs grow in the square root of the number of in-

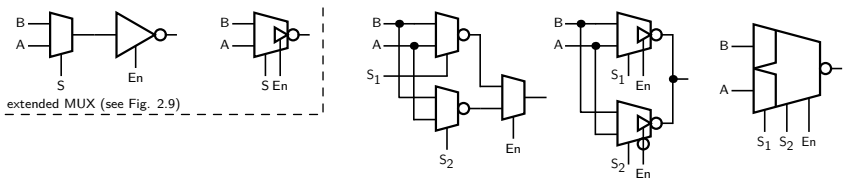


Figure 3.4: The extended multiplexer is equivalent to a MUX fed into a tri-state buffer and is depicted by a tri-state inverting multiplexer. The pattern of three multiplexers with only two distinct inputs, shown in the middle, is recurring in larger CCA circuits. Using MIGFETs, it can be compressed into a single-stage logic gate, depicted on the right. See Figure 2.9 for a circuit drawing explaining the principle operation.

puts (at least asymptotically). So, when the information density is thin enough, we should start to see gaps in the tree structure of the circuit.

In the CCA, one such gap shows up on the path to compute C_7. The relevant circuit pattern and its construction as a single gate is shown in Figure 3.4. Although three (more or less) independent signals select the input of the shown specialised multiplexer, only two inputs feed into the multiplexer instead of four, that it could theoretically select from. This obviously means that multiple selections map to the same input signal. But it also means that there is no need to keep all the necessary signal paths around for inputs that will never be used. The pattern in the middle shows clearly, that only one of the input multiplexers can be active at any given time. So, when the output of the inactive multiplexer could be tri-stated, they could be combined at the outputs without inferring another logic stage. (See second drawing from the right.) With MIGFETs such a circuit is possible and its construction is shown in the left box and in Figure 2.9 II. Adding one more gate to all transistors in the circuit allows us to also feed a third selection signal to the circuit, resulting in the final right logic gate, which shows a single-stage logic gate selecting over two inputs.

The utility of this circuit design lies its ability to exploit tri-state behaviour to connect the outputs of two multiplexers, which is usually forbidden in static logic design. Here, it does exactly allow to reflect the decision tree of the three selectors, which must be connected to increasing numbers of transistors in the logic gate depending on their level in the tree.

On a larger-scale arithmetic circuit, MIGFET implementations can fare quite well against CMOS circuits, as Table 3.4 shows. In the number of transistors, the MIGFET implementations range consistently in the 58 % region over all input sizes. The left number, in columns that show two numbers separated by a slash, represents circuits that do not replace the patterns of three multiplexers with the extended variant on the critical path, a constant difference of 6 transistors to the right number in those columns. The normalised delay for fanout $H = 4$ is shown in the next two columns. For the small 4-bit design, the introduction of 3-MIN logic gates dominates the improvements, pushing the delay down to almost

Table 3.4: Comparison of cca implementations for three input sizes. Normalized delay of the critical path. For the migfet implementations results are shown without / with the extended multiplexer gate.

Circuit	N_{FET}	N_{MIGFET}	$D^{H=4}_{\text{FET}}$	$D^{H=4}_{\text{MIGFET}}$
4-bit cca	144	82	43.6	26.2
8-bit cca	352	202 / 196	49.6	37.2 / 38.2
16-bit cca	826	480 / 474	68.5	51.9 / 52.5

1/2. Nevertheless, the larger the circuit gets, the more does the multiplexer tree dominate the circuit delay. Thus, the delay converges against 75 % of that of the cmos implementation.

In this chapter, we have seen that transistor reconfiguration and multiple gates per device are interesting opportunities to improve on logic circuit design. They show promising results for both explicitly reconfigured circuits in the reconfigurable architectures domain and implicitly reconfigured circuits as a general compression scheme that improves delay and energy performance. The effects are not only visible at a small scale but they carry over to larger circuit designs. In the next chapter, I will demonstrate an approach to model logic gates as transistor circuits which needs very little data. It targets early technology evaluation where reliable transistor device data is scarce, but where early results in circuit design can drive further device development. Thus, it focuses on standard-cell sized logic gates and enables us to automatically investigate implementation variants of Boolean functions and compare their results in various metrics and situations.

Chapter 4

Constructive DSE for Standard Cells Using Model Checking

The previous chapters showed that reconfigurable FETs have interesting new properties and lend themselves to new standard cell designs. I also showed that the enhanced features of the devices have an impact on larger circuits, as well, allowing for a more compact implementation of the intended function and a smaller energy budget. To this end, devices research has gone into the direction of polarity-controllable field-effect transistors based on established fabrication silicon processes [11, 64] or as low-temperature back-end processes using silicon and germanium that can be integrated into current fabrication processes as shown in [23, 56]. Other devices, such as carbon nanotubes [6, 37], graphene nanoribbon-based devices [15, 44], and tunnelling devices for ternary logic [26] have been proposed as well. To stay consistent with the earlier chapters, this chapter makes use of the germanium device shown in [56], as it shows promising performance results and TCAD simulation data was accessible for research during the work done resulting in this thesis. I will describe the modelling of the transistor using this particular device (see also Figure 2.6 for an impression of its performance data). Nevertheless, the modelling technique described here is general enough to describe any type of transistor device and several other device models were described with this approach, like a 400 nm silicon-based lab device described in [51] and a 32 nm high-performance FinFET device shown in [41]. When transistor devices with new properties are devised and published, it is worthwhile, if not important, to begin to evaluate their potential impact on circuit design as early as possible. Device and integrated circuit production is an excessively costly undertaking with many factors stacked against the adoption of new designs and stacked for being conservative and implementing small incremental changes. Stable device manufacturing, long-term working circuit designs are both in itself hard problems that take large amounts of time to be solved. Hence, when it is foreseeable that new device prop-

erties will influence very large-scale integrated (VLSI) design, one cannot start too early to try and quantify where circuit design should go forward with that new device or what properties of the new device may need special attention to make it competitive. Thus, providing an approach to explore the standard-cell design space and quantify key performance numbers like circuit delay, power dissipation, energy consumption per operation from as little device data as possible, was a key goal of this research. High automation ensures that, when device parameters can be determined with higher precision later, those changes are picked up on during quantification, allowing for a refined picture of the design space.

A second key goal of this research was to provide a tool that is capable of giving explicit answers to queries for extreme performance indicators where it made sense, like the absolute worst-case delay and maximum power dissipation; or to queries for average quantities, like average delay under the assumption of a certain reconfiguration probability or the average energy consumption per operation. This is a new direction of quantifying circuit designs early on, because it allows us to investigate variants of circuit designs which are functionally equivalent but whose performance differences are hard to judge from the outside. Having access to extremal values of quantities enables a better judgement of valuable implementation variants and allows us to explain *why* they are so valuable at the same time.

To achieve these two key goals, I chose to employ probabilistic model checking (PMC) as the computational basis to implement this automated design space exploration (DSE). Model checking has the prime feature that it provides guaranteed, exact and complete answers to queries that are checked against models of interest (see also [3], pp. 11 ff.). If a problem can be modelled and a meaningful query be formulated, the investigator does not need to have prior knowledge of corner cases or relevant input patterns. The approach in itself is able to determine the interesting model states and can even deliver them back to the investigator as witnesses. This is in stark contrast to simulation-based approaches which can only be as good as the quality of their test stimuli. Also, if model resolution affects the results, model checking will produce a new matching set of relevant states when model parameters change. In simulation, it is again left to the user to know about the changing conditions and to adapt the test set accordingly. Especially in variant analysis, where circuits are likely to have differing "critical" input patterns (those, that excite the worst-case result), and in early analysis, where device characteristics are likely to change, this independence of a well-known set of input stimuli is a complete game changer.

Up to this point, I completely ignored average performance metrics. These are next to impossible to obtain with simulations, but, on a large scale, are an even more important metric than worst-case metrics. The reason I believe this, is that large-scale systems all work asynchronously, anyways. So, in the long run, computer systems behave according to their long-run average characteristics, because that is the very definition of the long-run average of a metric. By induction, long-run average system performance is maximised when long-run average circuit

performance is maximised. So, when a certain worst-case performance is acceptable, aiming for the better long-run average performance may yield considerable benefits. This induction breaks down as soon as circuits are used in synchronous pipelined environments. Still, as the overall optimisation goal is still the one that is to be achieved (on the large scale), it would rather make sense to think about completely asynchronous (or rather self-synchronising) computer designs again, rather than dismissing the metric on the small scale (while it remains the only relevant metric on the large scale). It so happens that MIGFETs are also promising for use in asynchronous circuits [36, 13, 60] as they reduce transistor count and complexity (a main drawback of current designs) and their polarity control facilitates a direct implementation of hysteresis (a fundamental building block of self-synchronisation).

Standard cell characterisation can and is done in various ways. The method shown in [50] makes special effort to characterise reconfigurable standard cells to planar RFET technology. The fundamental difference of my proposed method to the established methods, FEM [52, 32] and SPICE [38], is that they describe a circuit experiment as an initial value problem of a set of differential equations that describe the model's evolution over time. The approach introduced in the following sections uses a charge transport model that operates with discrete voltages and evolves in time steps. State evolution is expressed through local interactions in which charges are transmitted between *charge stores* exclusively mediated by *charge transmitters*. The resulting models remain discrete and finite and can thus be completely traced and explored, which enables the exhaustive quantitative analysis using model checking. SPICE simulations are not generally feasible for exhaustive analysis and FEM, while it maintains the highest precision, becomes excessively costly for anything but the smallest circuits and the simplest experiments. The new approach allows me to directly query measures and perform multi-objective analysis, e. g. see [31, 21, 18].

Design space exploration of circuit variants from a single Boolean function is done via an algorithm that generates all "minimal" implementations – the minimality constraints are given in Section 4.4. This is not feasible at the transistor level using FEM, but in [66], a genetic algorithm found new multiplier circuits that were more compact and performed better than previously known variants, which could be analysed using SPICE.

Formal methods, in broader terms, have been used before in hardware verification and logic synthesis, see [29, 20]. Probabilistic model checking has been successfully applied to research the dependability of SRAM-based FPGAs in [24] and to quantify the reliability of NAND multiplexing in [39]. Theorem proving and model checking have been applied also to verify CMOS circuits regarding their conformance to a sequential specification and their functional correctness on a logic level [17]. Other approaches include the extension of very large-scale integrated circuit hardware description language (VHDL) and Verilog with domain-specific annotations that allow the automated extraction and verification of formal models, see [7, 19].

While timing analysis is usually done approximatively with gate-level simulations or static timing analysis [27], formal methods have also been successfully applied. As in this work, the idea was to overcome the incompleteness of approximation and the practicality issues that arise when attempting an exhaustive analysis with simulation-based techniques. To this end, probabilistic model checking could be used to verify complex RTL designs, like a H.264 decoder [30], and timed automata were used in [1], employing the model checker *Uppaal*, to analyse a 4-bit adder. Both works concentrated on evaluating gate-level designs constructed from standard cells, whose performance data had been obtained via other means. Using charge transport models, I can transition from an analogue circuit description to gate-level performance data, a technique that has also been adopted in SPICE in [59].

For the construction of an automated DSE tool, it was necessary for me to have a language that can efficiently capture this type of network model. The model checker PRISM, which was used to run the experiments, features a very basic input language. It is designed to exactly capture a probabilistic model description and provides no abstraction mechanisms to describe recurring parts of the model as functions and function calls. All connections must be described explicitly. This led to the development of `prism-gen`, a language that is based on CommonLisp and that allows the composition of transistors, sub-circuits and input stimuli generators with the surrounding test fixture, which is, then, compiled into a flat specification for PRISM. I introduce parts of the PRISM language and describe the unfolding of `prism-gen` constructs into it at appropriate times.

This chapter describes the principle operation of model checking, to enable the reader to understand the approach that I took to model and quantify reconfigurable circuit designs. It then goes on to describe the network model and the transistor model as well as their implementation as the `prism-gen` DSL. Lastly, this chapter focuses on the exploratory aspects and how implementation variants of a circuit are generated. The development of this approach is the result of a joint research effort with my colleague Steffen Märcker.

4.1 Principle Operation of Model Checking

Model checking is a *verification* technique. As such, it can determine the validity of properties with mathematical rigour. This is achieved by brute force analysis of all reachable states of the *model* under scrutiny. The model is a formalised representation of an actual computational problem – an algorithm, a computing device or a distributed system. It is restricted to just capture as much state of the real problem as necessary to faithfully replicate its behaviour when checked against a set of *requirements*. These requirements are also formalised in a property specification, where each property, again, captures the essential qualities that the model is to be proven (or disproven) to uphold. Thus, the mathematical proof, that model checking is able to deliver, exactly captures the two formalisations

(and simplifications) of the system to check and the property it shall possess, but not the real system and the real property themselves. As a verification method, it guarantees to "build the right thing" [3] (page 13). The biggest threat to its successful application is the *validation problem*, the question "do we verify the right thing?" Establishing the right model – a model that captures those qualities correctly that you want to base real-world decisions on – is an art. Inherent to the method of model checking, the user must supply formalisations of the system model and the properties that it is supposed to have. Both the types of models and the types of queries that the former can be checked against are interdependent. In this thesis, I exclusively use probabilistic model checking (PMC). This extension to model checking widens the applicability of the models to more interesting properties for which the model is best expressed as a probabilistic system or one that involves non-deterministic choice.

4.1.1 Model Types

Probabilistic model checking is used in this research to express the sets of random input stimuli to a circuit under test and it avoids picking (and potentially missing) relevant input stimuli that excite the necessary system behaviour that, in turn, shall be quantified as circuit delay or power dissipation. Finding the right input stimuli to excite maximum power dissipation or worst-case delay is taken from the hands of the developer and put into the hands of an automated tool. This greatly expands the usefulness of this formal method even beyond what is practical with simulation techniques.

Discrete-time Markov chain PMC requires the system model (i. e. the circuit itself and the whole experiment that surrounds it) to be expressed as a transition system that is augmented with probabilities and non-deterministic choices. This can be achieved in mainly two ways. As discrete-time Markov chains (DTMCs), the experiment is modelled such that all transitions from a state to its successor states are probabilistic. Thus, each state can be assigned a probability distribution according to which the experiment will select the next transition to its successor states.

Figure 4.1 shows an example of a DTMC. It is an input automaton that generates the stimuli for a digital circuit. The automaton describes the truth table over three inputs, thus, it has eight states. The transition to another state reflects the switch of one or more inputs. As the automaton shows, every state can only reach three neighbouring states, those, that reflect switching a single input. Furthermore, the graph can be divided into two major sub-graphs, of which one is outlined and labelled as "Configuration 0". Inside each sub-graph, state transitions are equally likely with a probability of 49 %. Transitions that are crossing the grey outline, though, are less likely to happen with only 2 %. This Markov chain depicts an input automaton for a 3-input reconfigurable circuit, where we assume that the first input is the reconfiguration input and the other two inputs

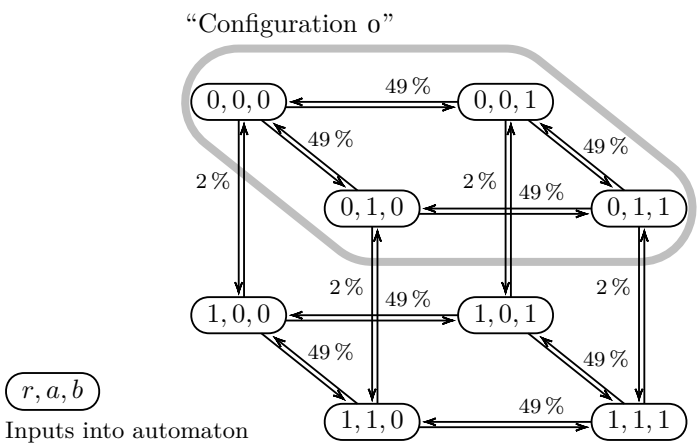


Figure 4.1: An input automaton into a 3-input digital circuit described as a DTMC. It can assume the eight states of the truth table and allows only single input switches. Input r is the reconfiguration input and the outlined Configuration 0 can only be left via switching that input. With 2 %, this is much less likely to occur than a switch of either input a or b (with 49 %).

are normal functional inputs. Thus, each sub-graph represents one of two configurations that the circuit would be in. We would use such an input automaton, for example, for experiments, where we would be interested in the average energy consumption per operation (for operations being restricted to single input switches) in which the circuit reconfiguration is not to be totally dismissed but its contribution being reduced to a realistic amount. Reconfiguration uses a lot of energy but is not very likely to happen. Regulating the switching probability of the reconfiguration input exactly achieves the desired effect. For the sake of simplicity, Figure 4.1 does not show the state space of the circuit model that is affected by this input automaton. Thus, it also does not show quantities on the transitions that, in the actual model, are the basis for its quantitative analysis (e. g. the absolute amount of energy per operation). This will be introduced later when describing the network model.

Markov decision process The second way to model a circuit is to describe it as a Markov decision process (MDP). MDPs allow the simultaneous use of non-deterministic choices and probabilistic ones. In general, this allows them to express a non-deterministic choice between probability distributions that govern the transition from a state to its successor state. From any state, first, a probability distribution is selected non-deterministically and second, the successor state is selected in a probabilistic manner like in DTMCs. Every DTMC is, thus, an MDP with only one probability distribution to choose from in each state.

Figure 4.2 shows a small non-deterministic automaton α and a possible MDP

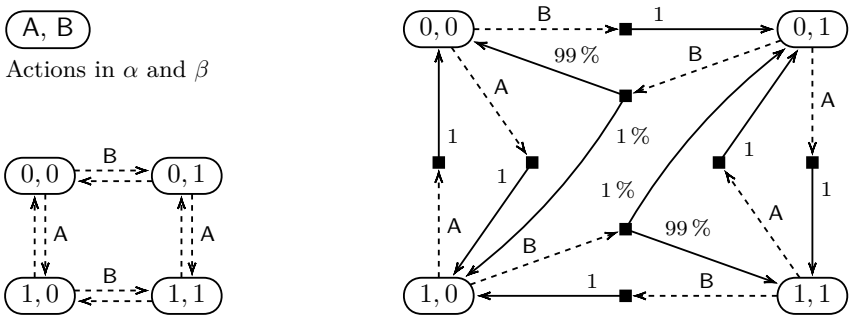


Figure 4.2: α. Another input automaton for a 2-input digital circuit with inputs A and B. In every state, the automaton selects one of two actions A and B. β. An MDP of automaton α, depicted as a bipartite graph. It introduces the possibility that a state transition fails. In each state, the system selects one of two actions corresponding to α. For both actions a probability distribution (black squares) shows the probabilities for a successful transaction or a failure; e. g. switching away with action B from states $(0, 1)$ or $(1, 0)$ fails in 1 % of the cases.

of that automaton as β. Taking a non-deterministic choice is called taking an *action* and is depicted by a dashed edge. Clockwise state changes are drawn using the outward edges and counter-clockwise changes use the inward edges in both graphs. So, in state $(0, 0)$ we can take action B to transition to state $(0, 1)$ or action A to transition to $(1, 0)$. This coincides with switching inputs A and B when the state encoding (a, b) encodes a as input A and b as input B. In the MDP β, model states are depicted as rounded rectangles, as before, but they are no longer directly connected. It is left by taking an action, which always selects a probability distribution (black squares). In this example, this almost always leads to choosing a probability distribution which only leads to a single outcome – the same target model state as in automaton α. For instance, leaving state $(0, 1)$ (top-right) by switching input A (downwards) leads to a square that has only one leaving probabilistic edge with probability 1 which leads to state $(1, 1)$ (bottom-right). This is true for all actions except for those leaving $(0, 1)$ or $(1, 0)$ via action B. With a probability of 1 %, the switch will fail and the model transitions to the state in the opposite corner. This corresponds to the behaviour that switching input B sometimes flips input A as well, but only if A and B start with opposite values. It could be modelling error behaviour that is expected due to manufacturing or technological constraints, but whatever it may model, it is clear that this behaviour affects the circuit differently than the probabilistic automaton in Figure 4.1. By exciting a switch in two inputs simultaneously, the circuit under test will exhibit drastically different timing and energy behaviour. This is something the example probabilistic automaton will not do.

While with DTMCs we could answer the query for the average circuit delay

or the circuit delay that is achieved in 99 % of the cases (under the given input distribution), there could always be a path from one model state to another that exhibits a longer circuit delay but would be very unlikely to be taken. So, this model would not deliver an absolute guarantee. MDPs deliver models that allow a different kind of properties to be checked. This model type allows us to query the absolute worst-case delay (or power dissipation) directly, and the model checker will also deliver a witness, which is a state (or list of states) that exhibits the queried behaviour.

Both model types can have designated initial states which, in general, must be carefully chosen, because they likely affect the states which can be reached in the model to a great extent. The model of an electrical circuit likely captures chaotic states that can never be reached by manipulating its inputs, e. g. interconnected network nodes that are charged to extremely different voltages. Thus, on the one hand, it is important to identify the relevant set of initial states to reduce the model size by removing well-known irrelevant states. On the other hand, these chaotic states could be interesting when investigating single-event upsets. Hence, the set of initial states is tied to the circumstances in which the properties that are checked also depend on the exact properties that are to be checked.

4.1.2 Query Types

The benefit of probabilistic model checking lies in its ability to return quantitative results directly, whose constructive parts are explicitly controlled during modelling. This enables its user to gain insights into properties, variations and trade-offs of implementations under scrutiny. The formalisation of both the model and the query enables a traceable and reproducible exploration that can operate fully automatic and without application-specific knowledge required from its user. In quantitative analysis using model checking one important decision to be made is which query language to use, and in this research my colleague Steffen Märcker and I opted for the use of CTL with the extension of probabilities and LTL path formulae. CTL sets itself apart from LTL in which the latter has a linear notion of *time* – that is a succession of states through the model – which means that its formulae uniquely describe linear paths starting from a state of interest. In CTL, branching can be considered in formulae which results in time being represented as infinite trees of states instead of sequences of states. These trees can be obtained by unravelling the model from a state of interest and recording all successor states in a (usually infinite) computation tree, hence the name *computation tree logic*. The general benefit of CTL over LTL is that there are fast algorithms known to compute the results. Also, the two logic systems are not directly comparable, i. e. one can express formulae in one that have no equivalent in the other. Hence, we use an extension to CTL, called CTL*, that allows us to use the expressiveness of LTL inside of CTL without immediately losing access to the fast algorithms to solve the formulae. Additionally, we use another extension of CTL which gives us access to reason about probabilities and expectations of paths through the model,

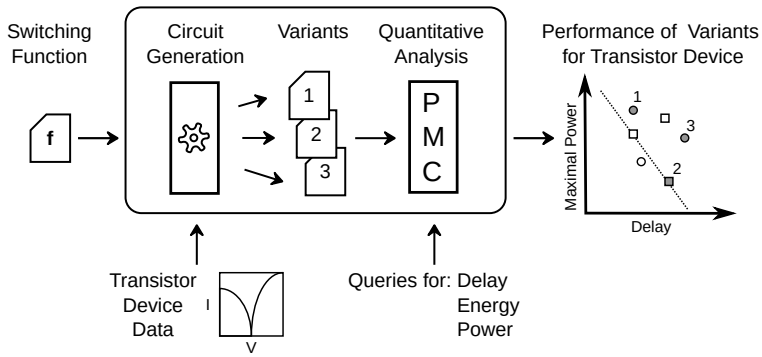


Figure 4.3: Overview of the design space exploration for a selected function and transistor.

PCTL*. I will not formally introduce CTL in this work as it would be of no benefit for the reader and rather refer to the original work of Clarke and Emerson [10] and the chapter on CTL in [3] pp. 313 ff. After introducing the transistor network models, I will pass along the queries that quantify certain behaviour only informally by using the relevant parts of the model.

4.2 Overview and Workflow

The automated quantitative analysis of circuits and circuit variants needs three fundamental user inputs 1) the Boolean function that is to be analysed, 2) transistor device data that can be used to derive transistor model and 3) queries that shall deliver interesting answers. Figure 4.3 shows how inputs 1) and 2) are taken up by the circuit generator to create circuit variants, which are successively analysed by the model checker according to the queries. I first demonstrated the working toolflow in [47].

The switching function is given as a Python Boolean function to the circuit generator which uses an adapted Quine-McCluskey algorithm to generate a set of variants of single-stage logic gates according to a set of minimisation criteria. It outputs the circuit variants as `prism-gen` files, one for each variant. These files do not describe a complete experiment – which would need a complete test fixture, describing the electrical surroundings like output loads, input transients' characteristics, the supply voltage and additional circuit parameters that should be quantified – but merely describe the electrical network models such that they can be included like library elements into a higher-level experiment model.

The transistor device data is not fed directly into the DSE but must first be converted into a transistor model as described in Section 4.3.3. Afterwards, it is, too, used as a library element in the process step of converting a particular circuit variant from a `prism-gen` model into a flat PRISM model.

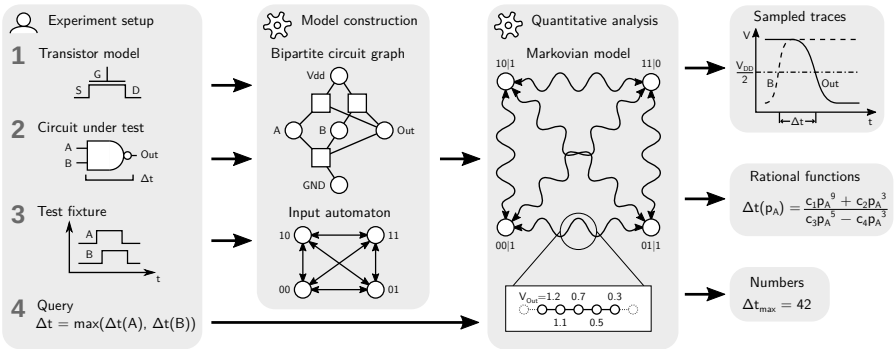


Figure 4.4: Detailed workflow of the quantitative analysis for a selected circuit and transistor.

4.2.1 Experiment setup

A more detailed workflow is shown in Figure 4.4 capturing the construction and analysis of a single circuit. The *experiment setup* encompasses all inputs that go into the analysis process.

1) Transistor model The designer needs at least one transistor model, which is usually taken from the library. Metal-oxide semiconductor transistors down to a gate length of about 10 nm can be captured by the transistor model shown in Section 4.3.3. Below that, the physics of conductance start to change again and other equations need to be used to capture the effects. In these cases, a new transistor can be instantiated by providing a parameter set instead of implementing a full model. The parameter set captures the following parameters for the whole device:

- $V_{\max}$ [V] the nominal device operating voltage,
- $C^{\mathrm{n}}_{\mathrm{chan}}, C^{\mathrm{p}}_{\mathrm{chan}}$ [F] the parasitic capacitance at the source / drain contacts,
- $\beta^{\mathrm{n}}, \beta^{\mathrm{p}}$ [S] the MOSFET equation's guidance value for n- and p-type conductance,

and for each transistor gate separately:

- V_{th} [V] the threshold voltage,
- $I^{\mathrm{NMOS}}_{\mathrm{th}}, I^{\mathrm{PMOS}}_{\mathrm{th}}$ [A] the drain currents at $V_{\mathrm{GS}} = V_{\mathrm{th}}$ and $V_{\mathrm{DS}} = V_{\max}$ for n- and p-type conductance,
- $SS^{\mathrm{n}}, SS^{\mathrm{p}}$ [V/dec] the sub-threshold slopes for n- and p-type conductance and

- C_{gate} [F] the gate capacitance.

The model can, then, be instantiated in the test fixture.

2) Circuit under test The circuit description may be the result of the circuit variant generator or a manual description in `prism-gen`, the details of which are introduced in Section 4.3. To allow maximum flexibility and code re-use, I separated the circuit network description from the definition of the test fixture and transistor. Although not a necessity, the circuit description is usually kept in a separate file and loaded into the test fixture. The electrical network is described as a bipartite circuit graph and is exemplified in the *Model construction* box in Figure 4.4 for a 2-NAND circuit (cf. Fig. 2.10 I). It is explained in detail in Section 4.3.2. This description is a circuit blueprint, and the actual transistors are only applied in the test fixture. Thus, the same circuit model can be used to quantify the design for different transistors. It is also used to describe a generic inverter that is instantiated multiple times in the test fixture to generate technology-specific input signals from discrete pulses.

3) Test fixture The test fixture is the top-level specification, which binds together the domain-specific aspects of the experiment, like, which circuit and transistor to use, with the modelling and model checking aspects, i. e. what values to quantify (and how), the input automaton, model initialisation and more. Listing 4.1 shows the general structure of a `prism-gen` input file. It displays the run-down of a test fixture description showing the functional parts in the commentary without going into detail and includes simple examples. I refer the reader to the Appendix on page 129 f. for a detailed description of the symbols and font key used to explain the syntax of `prism-gen`.

Each `prism-gen` input file consists of a single top-level form of the form `(net ...)` that encloses the content, regardless of it describing a transistor network or a device specification or, as in this case, a test fixture. The *`identifier`* is purely descriptive and has no functional meaning, but the `(:type ...)` argument must be set to generate either a DTMC or MDP model as this influences the type of state transitions and queries that can be used to build and check the model.

Usually, the first block of statements involve defining the primary model constants. The three constants `V_SCALE`, `T_SCALE` and `V_MAX` must always be defined, because other internal constructs for creating the input stimuli generator depend on their existence. Yet, this is left to the user to allow them maximum freedom in defining those constants in terms of other higher level constants or to express relations between them. Especially for checking probabilistic queries, it can be useful to express the test fixture functionally and leave the actual input probabilities undefined. This is exemplified in Line 12 with the constant `p_B` which is defined as the complementary probability to `p_A`, which is left undefined. Thus, the actual input probabilities can be shifted when running the experiment without

Listing 4.1: Abstract example of a test fixture in `prism-gen`.

```
(net identifier (:type pmc-model-type)

  ;; a set of model constants, e.g.
  ;; − voltage resolution, time resolution etc.
  (constant identifier type [value])
  (constant V_SCALE int)

  ;; experiment parameters, e.g.
  ;; − output load factor
  ;; − input switching probabilities etc.
  (constant p_A double)
  (constant p_B double (- 1 p_A))

  ;; circuit terminals: outputs, inputs, power supply
  (out-term circuit-output load-factor)
  (plus Vdd-identifier :value Vdd-voltage)
  (minus Vss-identifier :value Vss-voltage)
  (input-node input-identifier :slope input-slope-timesteps)

  ;; input automaton
  (input ...)

  ;; transistor model
  (load transistor-model-file library-prefix)

  ;; input signal generation

  ;; circuit instantiation
  (load circuit-file identifier :nodes ({terminal}+
                                        {transistor-type}+))
  ;; model initialisation, i.e.
  ;; initial values for all charge storage nodes and input signals
  (init :vars ...)

  ;; additional model computations, e.g.
  ;; − computation of specific transient crossing points,
  ;; − hazard conditions etc.

  ;; rewards definitions, i.e.
  ;; − defining which model values are quantised how
  (rewards reward (condition input-variable))
  (rewards V_output (T V_output)))
```

changing the test fixture; yet, the constraint that all input probabilities amount to 1 is maintained through the functional description.

The test fixture *grounds* the circuit network and defines all electrical reference points, the inputs, outputs and the supply and ground terminals. Using the `(plus ...)` and `(minus ...)` statements the designer establishes the logical and electrical relations of the supply and ground terminals with the rest of the network. They are explained in detail in Section 4.3.2. The test fixture defines the input automaton, which generates the digital input stimuli which are, then, translated into electrical stimuli via modelled inverters.

Both the transistor model and the circuit network are loaded as library elements via the `load` statement. Depending on use, this statement accepts a `:nodes` argument to parametrise the model and connect its terminals to the surrounding network. In contrast to simulation, model checking needs an explicit model initialisation. Simulation always follows a single trace exploring one timeline of model evolution along its simulation run. Model checking explores all possible model evaluations simultaneously. Thus, not initialising a model means that the model checker is expected to explore all evaluations from all possible starting states. This is usually not what we expect from an electrical test fixture. Hence, the designer defines all voltages at all points in the electrical network as well as the initial digital input signals.

Some experiments may involve computations which require additional knowledge about the circuit and the query that it is checked against. Thus, there is room to describe arbitrary computation and model extensions in a test fixture file. Lastly, the test fixture defines *rewards*, which are a model checking term for quantities. Listing 4.1 defines the output voltage as quantity of interest in Line 42, which, as a fundamental unit of the network model, is trivially defined as the voltage from the output terminal called `output`.

4) Query In the approach presented here, queries formalise the goal function that shall be computed in an experiment. These can be rather abstract questions to the test fixture, e. g. a performance measure such as circuit delay and energy consumption, but also quality measures such as functional correctness or the presence of output hazards. In addition, quantity measures can be related to the stimuli generated by the input automaton, to characterise the long-run behaviour of a circuit. Queries represent a generic characterisation of the set of traces that have to be taken into account. Model checking is, then, able to reason about the possibly infinite set of all traces that meet the criteria laid out in the query without actually generating them. This expressive power is very different from simulation-based approaches which rely on an explicit enumeration of the relevant traces.

Queries are a direct input to the model checker and do not influence the construction of the circuit model or test fixture. This makes them independent of the circuit implementation and thus, they are provided as a library to pick from.

Nevertheless, formalisation requires the designer to specify relevant model states in generic terms and to impose constraints on the temporal sequence in which they may occur in an experiment. The query, as the second input into the model checker next to the model, shares a formal relation with the test fixture. Not all types of models and queries are compatible with each other, e. g. PCTL* queries do not apply to transition systems with non-deterministic choices, generated by MDP models, and will generate an error when used together.

The experiments in this thesis will feature queries capable of determining the following results under various conditions:

- the worst-case input / output delay, power dissipation and energy consumption (per operation)
- the long-run average input / output delay and energy consumption (per operation)
- output hazards, i. e. spurious output switching events under the condition that an input switch does not finally result in an output switch

I will explain their structure and relations to the results informally in Section 4.3.4. The actual queries used will not be explained, because it is of little value to the reader and would involve a thorough introduction into temporal logic which is out of the scope of this document.

4.2.2 Quantitative Analysis and Results

The *model construction* and *quantitative analysis* are completely automated steps of the workflow. It is sufficient to describe a set of experiments as shown in Figure 4.3, which includes unmentioned details about specifying which models to analyse with which queries under what experiment conditions. I provide tools to start the whole process and collect the results as comma-separated values (CSV) files. These include unfolding the experiment description into individual scripts and command files for easy distribution of the quantitative analysis over a range of compute servers and collection of the results.

The types of results can be numbers and Boolean results, which represent quantitative and qualitative results. Further, they can be rational functions that describe the probabilistic behaviour parametrically. Their particular value lies in their ability to describe a general circuit behaviour over a range of input conditions, which can be later used to assess circuit behaviour under concrete environmental conditions without the need to re-run the expensive process of model checking again. I use this in this thesis to present a solution to the following questions: For a set of implementation variants of a single Boolean function, whose input / output delays varies over a range of input signal distributions. Which circuit performs best under which input signal distribution and how good. The result will be a set of rational functions which cross each other at certain input signal distributions.

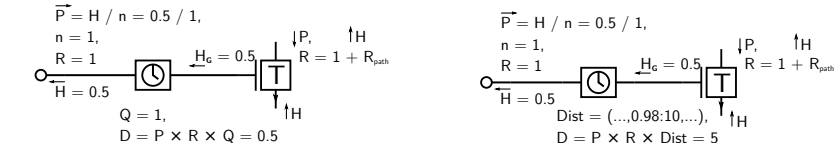
Results can also come in form of witnesses, which are complete model state descriptions that cause a query to apply. This means there are no probabilistic or non-deterministic choices left and the model deterministically evolves along a path that is captured by said query. A designer can use these witnesses to explore secondary effects of circuit states, which are not captured by the query and thus not quantified. Many details of a circuit model, like the voltage at certain nodes which are not the circuit outputs or currents flowing through individual transistors are not returned by the abstract queries which determine related quantities like power or energy. Nevertheless, for the understanding of a particular circuit behaviour, it might still be interesting to look at all the electrically interesting properties when encountering a peculiar phenomenon through a query. Witnesses deliver this "back reference", which allows a designer to jump back to a model state which will result in the queried one, and then perform a simulation considering various other model properties after the fact. This allows additional insight into why a circuit exhibits a certain behaviour.

4.3 Transistor Circuit Model

This section describes the model that drives this research and produced the results that were shown in earlier and, especially, in the next chapters. The implementation of this model was done in the domain-specific language `prism-gen`, which was specifically designed for conducting this investigation. It was necessary to create a new language, because, while the model checker PRISM was the only available tool able to perform the necessary model checking, it lacks the abstraction to supply a high-level specification of an electrical circuit as a computable model. `prism-gen` is the entry language and a transpiler that made this research possible. In the following sections, I will interleave the presentation of the formal conception and practical implementation of the components of the transistor circuit model.

4.3.1 Direct Logic Network Model

It is the scope of this research to investigate and model the behaviour of reconfigurable transistor circuits with the focus on complementary static logic gates. Thus, in a first iteration, I considered an abstract transistor device model that implemented a six-valued logic of strong and weak binary signals as well as tri-state and collision signals. It was clear from the beginning, that the reconfigurable transistor would be driven by two independent gate inputs, which, in conjunction, would drive and configure the device and that only three of the possible four states that were encoded by its two inputs would result in a behaviour that would directly map to Boolean logic. The fourth state would excite ambipolar behaviour, and its influence on surrounding transistors would depend on their state as well as the ambipolar one (cf. Section 2.2.2). Thus, it was necessary to use a logic with more than two values. In addition to the value domain, I abstracted



α. β.

Figure 4.5: Direct logic transistor delay models. α. an exponentially distributed delay around expectation $Q = 1$. β. a counting channel with probabilistically selected delay values according to distribution Dist.

the time domain, i. e. switching delay, by an internal clock with an exponential distribution around a mean switching time as shown in Figure 4.5 α.

The transistor model in α switches around an expected value Q, but can switch a little early or late. The transistor model uses P as the dynamic load which describes how much suppliers n must drive the load factor H, which accumulates over branches, as:

$$P = \frac{H}{n}$$

Additionally, a resistance R slows down signals which are sent through transistor channels and which accumulate to a path resistance R_{path}. Thus, the transistor delay D is computed as:

$$D = P \times R \times Ex$$

where Ex is the expected delay, which comes from Q or Dist, depending on whether delay model α or β is used. Communication delays are neglected but could be easily considered by adding two-terminal devices which possess a clock with an expected switching time that reflects the expected communication delay. The technique is commonly known as a *digital clock* and has been successfully used in more abstract circuit delay models that work with whole logic gates or even larger sub-circuits. Although there are proven tools to describe and generate models using digital clocks, I implemented both variants α and β, because it was not clear from the beginning which model, if any, would yield a usable representation.

The idea is that the mean switching times could be randomized around a mean general switching time, to reflect production variation. Also, inputs are selected randomly or probabilistically to excite the interesting circuit states. The clocks of those transistors, that would produce the output value, would be modelled such that their delay value reflected the output load as well as the circuit structure. All this means, that the timing model (what logic value is output after what delay) and the value model (which logic value is actually output) are two separate models blended together, where the timing model influenced intermediate states in the value domain.

The value of this model is that satisfiability, i. e. whether a transistor network exactly performed an intended Boolean function, can be directly answered. This

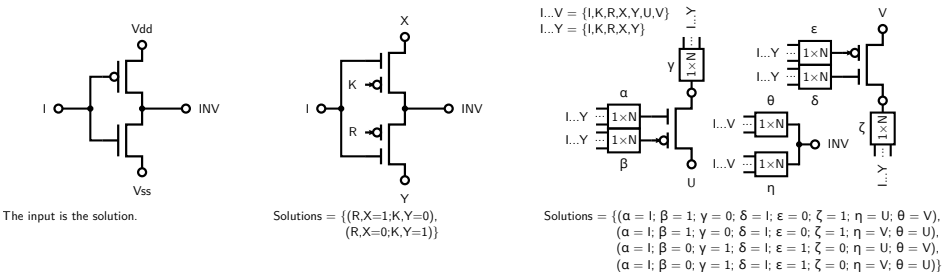


Figure 4.6: Direct computation of all valid inverter implementations for three models. *Left* a trivially correct CMOS implementation. *Centre* an RFET implementation with model variables K, R, X and Y and two solutions. *Right* a sea-of-gates model (cf. Sec. 2.1.1) with three components, two transistors and a join node, seven model variables and four solutions.

means that there is a road to use the model checker to build meta-circuits as shown in Figure 4.6 on the right. They capture a number of transistors and potential connections in a sea-of-transistors. The elements are initially independent from another and establishing connections between them is the task of the model checker. Each output connector is labelled with a single variable, e. g. U and V, and can act as an additional input to one of the input connectors. Each element has one or more input connectors that resemble switch-boxes. The input connectors, labelled with small Greek letters α to θ, can select from a variety of signals. Solutions are computed by checking these models against the goal logic function. The four solutions to the right meta-circuit for the logic inversion function are shown in Figure 4.6. They differ in their selection for the input pairs γ, β and ζ, ε, which program the respective transistors to either PMOS or NMOS function. By fixing the source contacts to opposing constant inputs – one transistor is fixed to 0 or a selection of variables, the other is fixed to 1 or a selection of variables – the designer can describe the expected complementary layout and avoid needless redundant solutions without cutting off real solutions. Additionally, at the merge node, the inputs η, θ are ordered, which means both solutions are doubled by wiring the transistor outputs U and V to input η and θ, respectively. This can also be addressed by removing one of U and V from the selection space of inputs η and θ. Limiting the input space by these two means can be achieved with only local knowledge (of the function of the merge node or the transistor node), which means that identifying these simplifications is independent of the goal function and does not grow in complexity with more complex meta-circuit models.

Direct logic models can also serve for quantitative analysis, as the direct logic transistor delay models from Figure 4.5 can be integrated into the sea-of-gates models. Thus, additional constraints enable model checking to find not only all functionally correct configurations but also the fastest designs.

A third major benefit of a direct logic model is that by staying in the same value

domain for functional as well as quantitative purposes, a quantified circuit could be abstracted into a much simpler model that consisted of a single logic function and a set of delay channels that model its input / output delay. Thus, not every component of more complex circuits needed to be modelled and quantified at the transistor level of detail.

Yet, this model proved to be dysfunctional for several reasons. The exponential delay model from Figure 4.5 α allows switches that are arbitrarily close to the mean value but different from it (with arbitrarily small probabilities of those corner cases ever happening). This produces models with many edges which grow to an unmanageable size very quickly. The clocks in the circuit under test operate independently of each other; they are not influenced by neighbouring conditions but advance in no particular order. This means, that even if two transistors would switch at the exact same time, the model checker would, categorically, still order all switches that happen at that moment, even though the user is no longer interested in this order. Any Newtonian physical system will behave the same regardless of the order of events that happen at the same time. The mere existence of this order meant that all orders of states would have to be kept and considered during model checking, which exploded the state space with 10^{12} states for a 3-MIN circuit which was barely computable. 3-XOR was not computable at all, because no partial input selection would prematurely fix the output value, which meant that all possible states would have to be considered for all possible input combinations. Even though the number of edges could be reduced by employing the counter model from Figure 4.5 (when used with a simple-enough distribution), this still proved to be insufficient to reduce the network model to computable sizes.

Additionally, the restriction to a logic (even a 6-valued logic) proved to be too restrictive to capture the intricacies that are influence output behaviour when experiencing slow input transients. It turned out to be insufficient to model slow (shallow) input transients as delayed but prompt input changes. Lastly, the dynamic load P proved to be insufficient to capture electrical power or energy requirements. Thus, a separate power model independent of the delay model would have to be devised.

Although, a direct logic model has interesting properties from a circuit exploratory point of view, I did not pursue its implementation further and created a fundamentally different model.

4.3.2 Charge Transport Network Model

In the second attempt, I devised a model that directly operates in the analogue domain and developed that in a joint effort with Steffen Märcker. The idea of this model is to quantise the observable signals into voltage levels rather than logic values. Instead of capturing the value and the time domain in a single device, the network is separated into two categories of nodes and strictly laid out in a bipartite graph.

Each node in the network is either a *charge transport node* or a *charge storage*

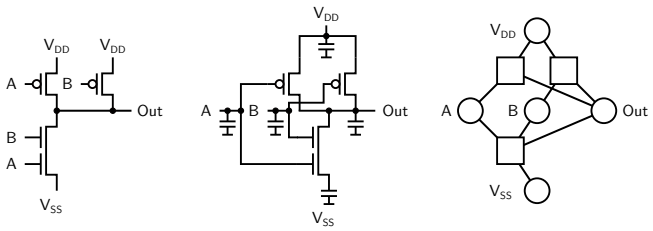


Figure 4.7: Translation of the CMOS 2-NAND circuit into a charge transport model. Transistors are split into functional devices and their surrounding capacitances. In the bipartite graph these are subsumed into charge transport nodes (squares), that carry the switching function, and charge storage nodes (circles), which capture the value domain.

node. Figure 4.7 shows a simplified translation of the well-known 2-NAND circuit into a charge transport model.

A *charge storage node* is a location in the network that has a definitive electric potential. In our model, these are absolute values and they are limited by the model parameters V_{scale} and V_{max} which describe the resolution and interval length of voltages that can be represented in the model. Because they are the only nodes that represent electric potential, they are also tied to a definitive (but not necessarily constant) capacitance which must be greater than zero. Thus, the value domain (voltage) and the timing domain (via capacitance) are tied together in these nodes in form of a virtual capacitance whose first terminal is connected to a Kirchhoff-node in an electrical network and whose second terminal is connected to ground (hence, the absolute voltage it represents). According to Kirchhoff's current laws, its capacitance is charged and discharged by the sum of all momentary currents through the attached nodes at that network location. All connected nodes are charge transport nodes due to the bipartite graph property.

A *charge transport node* characterises how charges are transported in terms of well-defined momentary currents flowing through all its connections. Thus, it must have two or more connections to surrounding charge storage nodes, which establish an absolute voltage potential on their particular edge. The resulting tensions are used, depending on the electrical device that a charge transport node represents, to calculate the momentary currents at all edges. In its simplest form it would resemble an idealised resistor and would possess a defined conductance. Its product with the voltage vector created by the two connected charge storage nodes would result in equal momentary currents, one flowing outwards and one inwards. Instead of full vector arithmetic, it was sufficient in this work to define the direction of the electrical current in relation to signed voltage differentials. On the influx side, the current is negative such that it draws charge from connected the storage node (in relation to its capacitance) and reduces its absolute voltage. Likewise on the efflux side, the current is positive and raises the absolute voltage of that storage node. Which side is influx or efflux is decided by the sign of the

voltage difference between the two connected charge storage nodes. It is the user's responsibility to define the relations of currents and voltages in a meaningful way. To improve the handling of circuit descriptions, these relations are not exposed directly but remain hidden inside high-level device models. The network model is as simple as that, which is why charge transport nodes usually carry all the electrical properties (including intrinsic capacitance) and roughly align with the "devices" in the network, while charge storage nodes can be viewed as the "connections" between the devices.

The formal specification of the charge transport model is that of a bipartite multigraph $N = (S, T, (X_t)_{t\in T}, E)$ where:

- S is a set of *charge storage nodes*,
- T is a set of *charge transport nodes*,
- $(X_t)_{t\in T}$ is a pairwise disjoint *family of contacts* at the charge transport nodes,
- $E\colon \bigcup_{t\in T} X_t \to S \times T$ is a set of *connections* that forms a mapping from contacts to edges between charge storage nodes and charge transport nodes such that for all $t \in T$ holds $c \in X_t$ iff $E(c) = (\cdot, t)$.

We describe a concrete model by composing charge storage and charge transport nodes according to these rules. They communicate their state via the edges using three functions that describe it in terms of the three aforementioned electrical properties:

$$\begin{array}{ll} \mathsf{C} : E \to C & \text{Capacitance in Farad (F),} \\ \mathsf{I} : E \to I & \text{Current in Ampere (A),} \\ \mathsf{V} : E \to V & \text{Voltage in Volt (V).} \end{array}$$

For the remainder of this section, we identify the function symbols C, I and V with their physical properties C, I and V. Functions are assembled piece by piece from partial formulae, as such, each storage node $s \in S$ provides voltage V_s, each charge transport node $t \in T$ contributes a capacitance (as a device property) C_c and a current I_c for all its contacts $c \in X_t$. Communication through network connections are spontaneous. So, for each connection $e = (s, t) \in E$, we define:

$$V(e) \stackrel{\text{def}}{=} V_s \qquad I(e) \stackrel{\text{def}}{=} I_c \qquad C(e) \stackrel{\text{def}}{=} C_c$$

In its current form, local capacitances are constant $C_c = \text{const}$, i. e. they do not change throughout the evolution of the network model over its state space. Local momentary currents I_c are stateless mappings involving momentary path conductance properties $G_{c,k}$ and voltages of immediate connections V_c, V_k with $c, k \in X_t$, such that:

$$I_c\colon G_{c,k} \times V_c \times V_k \to I_{c,k} \tag{4.1}$$

These mappings are what defines the electrical properties of a charge transport node and, thus, of transistor devices. Voltages computed in charge storage nodes are the only stateful elements of the electrical network. Their update function is defined as:

$$V_s \coloneqq \frac{\sum_{e \in E(e)} I(e)}{C_s^{\text{loc}} + \sum_{e \in E(e)} C(e)} \times t + V_s$$

with $E(e) = (s, \cdot)$ and C_s^{loc} an additional load attached to s.

Model time t and voltages V_s are discretised, which enables me to express the model with an enumerable set of states. Picking voltage over charge as the stateful variable, makes the expression of charge storage node automata independent of their connected charge transport nodes. The design of the network specifically targets digital logic circuits. Each point of the network operates at the same voltage levels, but depending on the network structure, vastly different amounts of charge may collect at the storage nodes. Thus, the network's state space becomes more predictable by predefining an amount of voltage levels than by computing the necessary amount of charge that a storage node needs to be able to hold. Additionally, varying capacitances (or negative momentary capacitance) would be hard to support when using charge as the state variable, because the number of states in a charge storage node needs to be constant.

This choice comes at the expense of loss of precision when handling nodes with extremely different capacitances in the same network. For large capacitances, voltage updates might be so small that they get rounded to 0, while for small capacitances, voltage updates might be so big that a lot of charge gets "lost" when the voltage reaches one of its limits (meaning the amount of charge in the network would not remain constant).

The model is discretised with two main scaling factors T_{scale}, which replaces t, and V_{scale}, which is used to express the update function in scaled integral units of voltage. Additionally, all components of the network work completely synchronous. Thus all voltage updates are computed atomically and rely only on the previous (atomically-generated) state. In this work, probabilism and nondeterminism exclusively comes from the input automaton. Although devices with probabilistic behaviour would be conceivable, my thesis does not target the investigation of device reliability or memory devices (e. g. floating-gate transistors), which would benefit from such extension. Nevertheless, the model is completely capable of supporting such behaviour, should the need arise.

Charge Storage Node

A charge storage node $s \in S$ holds its momentary voltage as the discretised value $D_s \in [D_{\text{SS}}, D_{\text{DD}}] \subset \mathbb{N}$. The boundary values D_{SS} and D_{DD} are obtained from the

three values V_{SS}, V_{DD} and V_{scale} by:

$$D_{\mathrm{SS}} = \lfloor V_{\mathrm{SS}} \times V_{\mathrm{scale}} \rfloor \qquad D_{\mathrm{DD}} = \left\lceil \max(1 + V_{\mathrm{SS}} \times V_{\mathrm{scale}}, V_{\mathrm{DD}} \times V_{\mathrm{scale}}) \right\rceil \quad (4.2)$$

Together with T_{scale} and T_{props}, the five values are the *references* to a charge storage node. The discretised voltage levels D_s are dimensionless, as I define V_{scale} as V^{-1}. The computation of D_{DD} ensures that $D_{\mathrm{SS}} \leq D_{\mathrm{DD}} - 1$, i. e. that there are at least two distinct discretised voltages of which D_{DD} is the larger one. Additionally, due to discretisation and the inexact double arithmetic, the computation of the momentary voltage of node s is actually performed in two steps. First, the voltage is computed as:

$$V_s = V_{\mathrm{SS}} + (V_{\mathrm{DD}} - V_{\mathrm{SS}}) \times \left(\frac{D_s}{D_{\mathrm{DD}} - D_{\mathrm{SS}}} \right)$$

Both the rounding directions for the limits D_{SS} and D_{DD} and the computation for the momentary voltage V_s require that $V_{\mathrm{SS}} < V_{\mathrm{DD}}$. Also from the calculation of V_s, it follows that a charge storage node reaches its lower limit at the reference V_{SS}. Second, the state update function, which actually works on discretised values, is rewritten as:

$$D_s \coloneqq \max(D_{\mathrm{SS}}, \min\left(D_{\mathrm{DD}}, D_s + \Delta D_s\right))$$

This ensures the limitation of D_s and the computation order – highlighted with parenthesis – minimizes rounding errors.

A charge storage node that does not change its state (is neither charged nor discharged), is considered *stable*. This allows a designer to trigger model state evolution depending on the stability of the electrical network. Also, this allows a designer to constrain queries by network stability, which allows them to include model states in quantification that cannot be identified by measures like voltage alone, i. e. delay quantification uses stability to distinguish intermediate output transients from the final, relevant ones. A charge storage node s is stable when:

$$\mathit{stable}_s \overset{\mathrm{def}}{\iff} \Delta D_s = 0 \quad (4.3)$$

In `prism-gen`, charge storage nodes are visible parts of the circuit network description. This means, the bipartite nature of the network is retained in the high-level description. In other definitions, they are subsumed by the "devices" in the network and are only indirectly represented via the connection specifications between devices. For experimentation purposes, I kept the storage nodes as visible parts, such that they can have additional properties assigned individually.

I distinguish three different types of charge storage nodes.

Voltage source These nodes provide a constant voltage and allow an arbitrary flux of charge:

$$V_s = V_s^{\mathrm{init}} \Rightarrow D_s = V_s^{\mathrm{init}} \times V_{\mathrm{scale}} \qquad \Delta D_s = 0$$

They are not influenced by capacitances from connected charge transport nodes and are defined as follows:

```
(net-node identifier [[ :nodes ({node-identifier}*) |
                        :input-voltage voltage |
                        :scaler factor-or-formula |
                        :digital-value logic-value ]]*)

(minus identifier [[ :nodes ({node-identifier}*) |
                     :references (V_SCALE) |
                     :value voltage |
                     :digital-value logic-value ]]*)

(plus identifier [[ :nodes ({node-identifier}*) |
                    :references (GROUND V_SCALE) |
                    :value voltage |
                    :digital-value logic-value ]]*)
```

Example:

```
;; register V_SS = 0 V and V_DD = 1.2 V nodes with a
;; discretisation of 10 000 steps/V (i.e. 100 µV)
(minus VSS :value 0 :references (10000))
(plus VDD :value 1.2 :references (VDD 10000))
```

All voltage sources use a defined voltage *voltage* in [V]. The general `net-node` needs a scaler, which is either a numeric factor which is multiplied with *voltage* or a formula which may contain the literal symbol `VOLTAGE` as a placeholder to the variable name that contains *voltage*. The `minus` and `plus` statements abstract this away to ensure that the condition $D_{\mathrm{SS}} \leq D_{\mathrm{DD}} - 1$ is never violated by computing the matching scaler internally based on the values for *V_SCALE* and, for `plus`, with and additional reference to *GROUND* according to Equation 4.2. Even if the rational value $V_{\mathrm{SS}} \geq V_{\mathrm{DD}}$, will the discretised value D_{DD} honour the condition. This ensures numerical stability of all dependent computations in the network model. `minus` and `plus` use the global symbols `V_SCALE` and `GROUND` as default references. Thus, when a designer names the V_{SS} node `GROUND` and defines a constant `V_SCALE`, the `:references` need not be specified when creating the voltage sources.

Voltage sources may also map a logic value to a voltage by providing and integer to the `:digital-value` argument. The `minus` and `plus` statements supply 0 and 1 by default. This is used in writing input automata for experiments, because in digital circuit design the automaton is usually easily defined in terms of Boolean formulae. Using integrals allows a designer to use a logic with more than two values, but computation in that logic would not be directly supported in either `prism-gen` or PRISM Model checker (PRISM) and would have to be simulated. Voltage sources are always considered stable.

The general `net-node` statement can be used to created other voltage sources of interest, e. g. to drive a circuit against a fixed voltage to excite certain analogue behaviour.

Dirac node These nodes, as their name suggests, work like a differentiator and excite a pulse depending on the value of a Boolean variable x. They output a constant voltage difference, i. e. describe a linear function, which is scaled by the factor T_{slope}:

$$\Delta D_s = \begin{cases} \Delta S_s & \text{if } x \\ -\Delta S_s & \text{otherwise} \end{cases} \qquad \text{where} \quad \Delta S_s = \left\lceil \frac{D_{\text{DD}} - D_{\text{SS}}}{T_{\text{slope}} \times T_{\text{scale}}} \right\rceil$$

T_{slope} describes the number of time steps required to perform a full output swing. By construction of the network model, state changes take at least one time step, which requires $T_{\text{slope}} \times T_{\text{scale}} \geq 1$.

These types of nodes are used to describe artificial input stimuli and are part of the first conversion step from a Boolean input value to an input transient that is realistic for a particular transistor technology. In `prism-gen`, these nodes are created with:

```
(input-node identifier [[ :nodes ({node-identifier}*) |
                          :references (SUPPLY GROUND V_SCALE
                                       T_SCALE T_PROPS) |
                          :slope attached-load |
                          :stable stablisation-formula ]]*)
```

Example:

```
;; create a slow (100 steps) and a fast input (1 step)
(input-node A :slope 100)
(input-node B :slope 1)
```

Again, when the voltage sources are named *`SUPPLY`* and *`GROUND`* and the other necessary constants are defined as well there is no need to explicitly specify the `:references` argument at creation. The `:slope` argument defaults to 1, and the `:stable` argument allows a designer to provide a different stabilisation formula. By default, the stabilisation criterion is redefined as:

$$\mathit{stable}_s \overset{\text{def}}{\iff} D_s = D_{\text{DD}} \vee D_s = D_{\text{SS}}$$

Another useful criterion, though, is to define it as $\top$ (`T` or `true` in `prism-gen`), which marks the node as always stable. Stability is used to determine whether the network is in equilibrium, which is used as a trigger to advance to the next input stimulus during the quantitative analysis. Removing a node from this consideration may be necessary to investigate overlapping input states, but must also used with caution, as marking too many nodes stable may result in nonsensical input patterns.

Integrator node The common nodes that represent the connections in a circuit diagram show integration behaviour, which is to be expected, because they are assigned a capacitance that is always greater than zero. They change their voltage according to Kirchhoff's current law based on the current via the connections to charge transport nodes. As voltages are discretised, the update is performed on the integral voltage D_s:

$$\Delta D_s = \mathrm{sgn}(\Delta S_s)\lfloor|\Delta S_s|\rfloor \quad \text{with} \quad \Delta S_s = \frac{I_s}{C_s} \times \frac{V_{\mathrm{scale}}}{T_{\mathrm{scale}}}$$

Rounding updates towards zero enhances numeric stability, as it avoids that very small residual updates, which are the result of discretisation errors, propagate through the network.

They are created as capacity network nodes:

```
(cap-net-node identifier [[ :nodes ({node-identifier}*) |
                            :references (SUPPLY GROUND
                                         V_SCALE T_SCALE) |
                            :load attached-load |
                            :stable stablisation-formula ]]*)
```

Example:

```
;; create inverter output node and connect drain sides of PMOS and NMOS
;; transistors
(cap-net-node inv_out :nodes (pmos_r nmos_r))
```

Each storage node has an identifier and takes up to four named arguments, none of which are mandatory. The most common argument is the specification of `:nodes (...)`, which directly connects the storage node to its relevant transport nodes. For special purposes, all model references can be redefined for this particular node. Due to the notion of charge transport nodes being the "devices" in the network, the device property *intrinsic capacitance* is usually kept with the charge transport node itself and communicated to the storage node when they are connected (for this to make sense, charge transport nodes of complex devices have individual capacitances on each connector). Thus, the capacitance of a charge storage node is the sum of the capacitances of all connected charge transport nodes. The `:load` argument adds the additional capacitance C_s^{loc} to the node on top of the connected transport nodes.

Charge Transport Node

The other type of nodes describes the electrical devices that react to the voltage differences that are created by connecting a number of charge storage nodes with each other. Generally, they are supposed to capture the three electrical properties capacitance, resistance and inductance. The last one is not currently implemented, as inductance does not play a major rôle in digital transistor circuits.

Inductance only has a negligible influence at the scale of standard cells. Also, the current design does not represent series capacitances directly as network nodes but only resistors. Nevertheless, those are captured by complex charge transport node definitions with more than two terminals.

All charge transport nodes can be subdivided into two types of components, a) the node itself that shall capture resistive and capacitive effects and b) its contacts, called terminals, that provide the connections to the surrounding charge storage nodes. So, for a node $t \in T$ with k contacts, there exists the index set $X_t = c_1, \ldots, c_k$.

Terminals in `prism-gen` provide a name under which the specific contact can be referenced and a capacitance towards the V_{SS} reference that they contribute to a connected charge storage node. A definition for a charge transport node $t \in T$ provides momentary currents I_c for each of its terminals $c \in X_t$ as shown in Equation 4.1 on page 70. Additionally, the momentary voltage V_c at the terminal $c \in X_t$ denotes the voltage $V(e)$ of the connection $e = (c, \cdot, t) \in E$. This makes it easier to correctly identify voltages as charge storage nodes are usually represented by anonymous wires in a circuit diagram, and it would be cumbersome and without merit to add φ symbols to denote voltages in all places.

Output terminal The simplest charge transport node is the output terminal, which is used to attach a load to a circuit output. It is defined as:

```
(out-term identifier [[ :load contributed-load |
                        :current drawn-current ]]*)
```

Example:

```
(out-term out :load 1e-11)
```

Its only contact is named after the terminal itself, i. e. `out` when the output terminal was created with the name `out`, as well. Additionally to specifying a contributed load, an output terminal allows a designer to use it as a current source by using the `:current` argument. So, in addition to capacitive loads, resistive loads can also be attached to the circuit under test to further evaluate its qualities.

RC circuit The smallest charge transport node that represents an electric "device" is the two-terminal RC circuit. It provides a series resistance and a capacitance towards V_{SS} on either end. RC circuits are defined as:

```
(rc-node identifier [[ :resistance series-resistance |
                       :load-l contributed-load-l |
                       :load-r contributed-load-r ]]*)
```

Example:

```
;; create an RC circuit R with contacts R_r and R_l
```

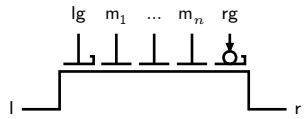


Figure 4.8: General mnemonic of the transistor model contacts. Conventionally, the right contact is facing the circuit output and the right gate describes the program gate. The exact meaning, though, depends on the parametrisation of the device, as it may be perfectly symmetric and either outer gate may serve as program gate.

```
(rc-node R :resistance 1e3 :load-l 1e-12 :load-r 1e-12)
```

Charge transport nodes inherently define the names of their contacts in relation to their own name by convention. They usually provide a *right* contact and a *left* contact named *name*`_r` and *name*`_l` respectively. These identifiers must be used when specifying connections to charge storage nodes. Also, the arguments to `rc-node`'s, or charge transport nodes in general, can be formulae that make use of the other defined parameters. This allows a designer to specify other behaviour like varistors and other two-terminal devices with variable resistance.

4.3.3 Transistor Model

The main charge transport node of this model is the transistor device. In the course of this work, compact models for emerging technology devices, like the polarity-controllable transistors shown throughout this thesis have not been developed, yet. Thus, in close collaboration with Steffen Märcker, I developed a new generic transistor model that is able to deliver results with very little parametrisation and which can use data published for single experimental devices. This model can be easily refined when more precise data becomes available, especially regarding unusual electric configurations, like ambipolar scenarios, and the automated analysis be re-run to update our research results. The model shown here is general enough to capture several emerging devices shown in [23, 57, 51]. I also use it to model a commodity MOSFET device [41] that serves as a comparison to the emerging devices for calibration purposes. The transistor model is purely functional, i. e. it does not increase the state space of the network model but computes electrical currents completely from the momentary voltage values of connected charge storage nodes.

In contrast to the common terminology used to describe unipolar MOSFET technology, reconfigurable transistor circuits are ambiguous in their use of *source* and *drain* contacts for transistors, especially because their polarity change would invert the naming of the contacts as well. This makes talking about the connections hard which is why I will use a terminology based on information flow throughout this section. Information flows from *left* to *right*, which means that a left transistor channel contact faces one of the supply voltage rails while the right

channel contact faces the output of the circuit as shown in Figure 4.8. By construction of polarity-controllable nanowire transistors, the polarity-control gate is placed near the output channel contact, which is why it is usually called the *right gate*. Likewise the transistor gate closest to the left channel contact is called the *left gate*. For reconfigurable nanowire transistors, these four contacts are mandatory, because channel doping is achieved electrostatically and not chemically via the right polarity-control gate while the left gate is used for steering the channel open and close. Nanowire transistors are able to host more than two gates as shown in [57, 51]. Transistor gates in-between the left and the right gate are called *inner gates*. They are counted from left to right as $\mathrm{m}_1, \ldots, \mathrm{m}_n$. A transistor with $n+2$ gates features the ordered set of gates $G = (\mathrm{lg}, \mathrm{m}_1, \ldots, \mathrm{m}_n, \mathrm{rg})$.

Due to the principle symmetry of the left and right transistor gates, this is covered by the equations in the transistor model. Thus, in the following description, I highlight which of the two outer gates is currently functioning as control gate (CG) and which as polarity control gate (PCG) (or program gate). All voltages are merely specified in relation to each other and named after their contacts as V_l, V_r, V_lg and V_rg. The four controlled states of the transistor which were introduced in Figure 2.5 on page 25 are *closed*, PMOS *open*, NMOS *open* and *ambipolar*.

The transistor model realises these modes according to the following inequalities:

PMOS Channel closed or open for h^+ charge carriers

$$\text{if} \quad V_\mathrm{l} > V_\mathrm{r} \geq V_\mathrm{rg} \quad (\mathrm{rg\ is\ PCG}) \qquad \text{or} \qquad V_\mathrm{r} > V_\mathrm{l} \geq V_\mathrm{lg} \quad (\mathrm{lg\ is\ PCG})$$

NMOS Channel closed or open for e^- charge carriers

$$\text{if} \quad V_\mathrm{l} < V_\mathrm{r} \leq V_\mathrm{rg} \quad (\mathrm{rg\ is\ PCG}) \qquad \text{or} \qquad V_\mathrm{r} < V_\mathrm{l} \leq V_\mathrm{lg} \quad (\mathrm{lg\ is\ PCG})$$

ambipolar Channel is open for h^+ and e^- charge carriers

$$\text{if} \quad V_\mathrm{l} > V_\mathrm{r} \text{ and } V_\mathrm{l} > V_\mathrm{lg} \text{ and } V_\mathrm{rg} > V_\mathrm{r} \qquad \text{or}$$

$$\text{if} \quad V_\mathrm{l} < V_\mathrm{r} \text{ and } V_\mathrm{l} < V_\mathrm{lg} \text{ and } V_\mathrm{rg} < V_\mathrm{r}$$

In PMOS and NMOS configurations the left inequalities are configurations in which the left gate works as the control gate steering the channel, while the right gate does polarity control. In the *ambipolar* configuration, their rôles are ambiguous and turning off either gate results in the transistor assuming the open states in either PMOS or NMOS configuration (cf. Figure 2.5).

The transistor models in this work assume a negligible gate leakage $I_g = 0\,\mathrm{A}$, because reliable data is not available and gate leakage is known to be at least an order of magnitude lower than channel leakage. Yet, channel leakage is also neglected in this work, because simulations showed a very high ratio of $I_\mathrm{on}/I_\mathrm{off}$ currents with seven orders magnitude for low power devices. I_off currents were so small that reliable simulation or test data was not available as of this writing. Due to the directional definition of the transistor model as a charge transport

node with a left and a right side, the channel currents I_l and I_r are defined in terms of a directed inner drain current I_D as follows:

$$I_\mathrm{l} = \begin{cases} -I_\mathrm{D} & \text{if } V_\mathrm{l} > V_\mathrm{r} \\ I_\mathrm{D} & \text{otherwise} \end{cases} \quad \text{and} \quad I_\mathrm{r} = \begin{cases} I_\mathrm{D} & \text{if } V_\mathrm{l} > V_\mathrm{r} \\ -I_\mathrm{D} & \text{otherwise} \end{cases}$$

The transistor is modelled as a piecewise combination of functions which model the various aspects of its geometry and function. It is operated in *directional modes* which govern the direction of its inner drain current I_D while the value of that current is the superposition of partial behaviour influenced by its *multiple independent gates*.

Directional modes While the direction of I_D is governed by the difference between V_l and V_r, the device does not behave linear with respect to them but operates in three different *directional modes*. The first two, "left open" and "right open", are the modes of a properly programmed transistor, where the left inequality describes p-channel polarity and the right inequality describes n-channel polarity.

$\mathrm{l} \to \mathrm{r}$ (Left open): Charge carriers can enter the channel from the left. In this work, contact l is called the source, r is drain, the right gate rg acts as program gate (PCG) and the following holds:

$$\forall g \in G.V_g < V_\mathrm{l} \qquad \text{or} \qquad \forall g \in G.V_g > V_\mathrm{l}$$

$\mathrm{l} \leftarrow \mathrm{r}$ (Right open): Charge carriers can enter the channel from the left. Contact r is the source, l is drain, the left gate lg acts as PCG and it holds:

$$\forall g \in G.V_g < V_\mathrm{r} \qquad \text{or} \qquad \forall g \in G.V_g > V_\mathrm{r}$$

$\mathrm{l} \leftrightarrow \mathrm{r}$ (Both open): Charge carriers can enter the channel from both sides. The device is ambipolar and both Schottky barrier gates steer the channel open. It holds:

$$V_\mathrm{l} > V_\mathrm{lg} \text{ and } V_\mathrm{r} < V_\mathrm{rg} \qquad \text{or} \qquad V_\mathrm{l} < V_\mathrm{lg} \text{ and } V_\mathrm{r} > V_\mathrm{rg}$$

These modes are not exclusive and according to published data in [23, 58] they can be approximated as the maximum of the three currents calculated according to the directional modes:

$$I_\mathrm{D}(\mathbf{V}) \stackrel{\text{def}}{=} \begin{cases} \max(I_{\mathrm{l}\to\mathrm{r}}(\mathbf{V}), I_{\mathrm{l}\leftarrow\mathrm{r}}(\mathbf{V}), I_{\mathrm{l}\leftrightarrow\mathrm{r}}(\mathbf{V})) & \text{if } V_\mathrm{DS} > 0\,\mathrm{V} \\ 0\,\mathrm{A} & \text{otherwise} \end{cases} \tag{4.4}$$

with $V_\mathrm{DS} = |V_\mathrm{l} - V_\mathrm{r}|$ and $\mathbf{V} = (V_c)_{c \in X_t}$ the vector of voltages for all transistor gates for a transistor t.

Multiple independent gates In the modelled transistor, all transistor gates except the drain-side gate act as control gates in the *left open* and *right open* directional modes. They operate as a wired-AND, which means a single gate can close the channel while all control gates have to agree to open it. Available TCAD simulation data shows that in the *both open* directional mode, only the Schottky barrier gates control the channel. By closing off either side, the transistor reverts to either the *left open* or *right open* directional mode. The data also shows that the control gates restrict the current through the transistor channel resulting in the minimum electrical current being effective in those directional modes. The threshold voltages $V_{\mathrm{th},g}$ are modelled individually per transistor gate $g \in G$ and projected into the voltage differences between the gate and the channel contact $c \in \{l, r\}$ as $V_{g,c} = |V_c - V_g| - V_{\mathrm{th},g}$. The directional currents are, then, defined as:

$$I_{\mathrm{l}\to\mathrm{r}}(\mathbf{V}) \stackrel{\text{def}}{=} \begin{cases} \min\left\{ I^{\mathrm{PMOS}}_{\mathrm{D},g}(V_{\mathrm{DS}}, V_{g,\mathrm{l}}) \colon g \in G \right\} & \text{if } \forall g \in G.V_g < V_{\mathrm{l}} \\ \min\left\{ I^{\mathrm{NMOS}}_{\mathrm{D},g}(V_{\mathrm{DS}}, V_{g,\mathrm{l}}) \colon g \in G \right\} & \text{if } \forall g \in G.V_g > V_{\mathrm{l}} \\ 0\,\mathrm{A} & \text{otherwise} \end{cases}$$

$$I_{\mathrm{l}\leftarrow\mathrm{r}}(\mathbf{V}) \stackrel{\text{def}}{=} \begin{cases} \min\left\{ I^{\mathrm{PMOS}}_{\mathrm{D},g}(V_{\mathrm{DS}}, V_{g,\mathrm{r}}) \colon g \in G \right\} & \text{if } \forall g \in G.V_g < V_{\mathrm{r}} \\ \min\left\{ I^{\mathrm{NMOS}}_{\mathrm{D},g}(V_{\mathrm{DS}}, V_{g,\mathrm{r}}) \colon g \in G \right\} & \text{if } \forall g \in G.V_g > V_{\mathrm{r}} \\ 0\,\mathrm{A} & \text{otherwise} \end{cases}$$

$$I_{\mathrm{l}\leftrightarrow\mathrm{r}}(\mathbf{V}) \stackrel{\text{def}}{=} \begin{cases} \min\left\{ I^{\mathrm{PMOS}}_{\mathrm{D,lg}}(V_{\mathrm{DS}}, V_{\mathrm{lg,l}}), I^{\mathrm{NMOS}}_{\mathrm{D,rg}}(V_{\mathrm{DS}}, V_{\mathrm{rg,r}}) \right\} & \text{if } (V_{\mathrm{l}} > V_{\mathrm{lg}}) \wedge (V_{\mathrm{r}} < V_{\mathrm{rg}}) \\ \min\left\{ I^{\mathrm{NMOS}}_{\mathrm{D,lg}}(V_{\mathrm{DS}}, V_{\mathrm{lg,l}}), I^{\mathrm{PMOS}}_{\mathrm{D,rg}}(V_{\mathrm{DS}}, V_{\mathrm{rg,r}}) \right\} & \text{if } (V_{\mathrm{l}} < V_{\mathrm{lg}}) \wedge (V_{\mathrm{r}} > V_{\mathrm{rg}}) \\ 0\,\mathrm{A} & \text{otherwise} \end{cases}$$

Single gate The drain current for an assumed single gate transistor is modelled after the standard MOSFET equations [54], which describe it as a piecewise function in the gate voltage $V_{g,c}$ and the source-drain voltage V_{DS} with three non-overlapping intervals:

$$\begin{aligned} \text{Sub-threshold}\colon\ & V_{g,c} \leq 0\,\mathrm{V} \leq V_{\mathrm{DS}} \\ \text{Saturation}\colon\ & 0\,\mathrm{V} < V_{g,c} \leq V_{\mathrm{DS}} \\ \text{Linear}\colon\ & 0\,\mathrm{V} \leq V_{\mathrm{DS}} < V_{g,c} \end{aligned}$$

The drain current for every single gate $g \in G$ (neglecting all other gates) and a specific channel polarity $\text{POL} \in \{\text{NMOS}, \text{PMOS}\}$, $I^{\text{POL}}_{\text{D},g}$, is constructed from:

$$I^{\text{POL}}_{\text{D},g}(V_{\text{DS}}, V_{g,c}) \stackrel{\text{def}}{\equiv} \begin{cases} I^{\text{POL}}_{\text{D0},g}(V_{\text{DS}}) \times \left(SS^{\text{POL}}_g\right)^{V_{g,c}} & \text{Sub-threshold} \\ I^{\text{POL}}_{\text{D0},g}(V_{\text{DS}}) + \frac{\delta^{\text{POL}}_g}{2} \times V^2_{g,c} & \text{Saturation} \\ I^{\text{POL}}_{\text{D0},g}(V_{\text{DS}}) + \delta^{\text{POL}}_g \times \left(V_{g,c} \times V_{\text{DS}} - \frac{V^2_{\text{DS}}}{2}\right) & \text{Linear} \end{cases}$$

$$\text{with} \quad I^{\text{POL}}_{\text{D0},g}(V_{\text{DS}}) \stackrel{\text{def}}{\equiv} I^{\text{POL}}_{\text{th},g} \times \frac{V_{\text{DS}}}{V_{\text{DD}}}$$

This construction deviates from the standard equations in various aspects. First, the partial function describing the sub-threshold behaviour does not usually depend on V_{DS}. Yet, I scaled the drain current at the threshold voltage $V_{\text{th},g}$, $I^{\text{POL}}_{\text{th},g}$, with V_{DS} to compensate for measured effects that don't concur with the standard equations. Second, by using the scaled threshold current $I^{\text{POL}}_{\text{D0},g}$, all three pieces of the function meet in $(V_{\text{th},g}, I^{\text{POL}}_{\text{D0},g})$, which mitigates a jump in the drain current that is exhibited while crossing the threshold voltage using the standard MOSFET equations.

These equations depend on the momentary gate voltages $\mathbf{V}$ and a set of measured (or otherwise established) parameters that characterise the isolated influence of a single transistor gate on the drain current in either channel polarity: the threshold voltage $V_{\text{th},g}$, the threshold current $I^{\text{POL}}_{\text{th},g}$, the off-current $I^{\text{POL}}_{\text{off},g}$ and the gate capacitance C_{gate}. I define the *conductance per voltage* δ^{POL}_g, which is related to the usual MOSFET conductance expressed as β but scaled to fit the adapted model, and the *sub-threshold slope* SS^{POL}_g as follows:

$$\delta^{\text{POL}}_g \stackrel{\text{def}}{\equiv} 2 \times \frac{I^{\text{POL}}_{\text{on}} - I^{\text{POL}}_{\text{th},g}}{\left(V_{\text{DD}} - V_{\text{th},g}\right)^2} \qquad \text{and} \qquad SS^{\text{POL}}_g \stackrel{\text{def}}{\equiv} \left(\frac{I^{\text{POL}}_{\text{th},g}}{I^{\text{POL}}_{\text{off},g}}\right)^{\frac{1}{V_{\text{th},g}}}$$

The on-current $I^{\text{POL}}_{\text{on}}$ is ultimately limited by the Schottky barriers and, thus, independent of a particular gate. It is the drain current that is observed for a fully opened transistor and $V_{\text{DS}} = V_{\text{DD}}$. The equations computing I_{l} and I_{r} via I_{D} implement Equation 4.1 and establish compatibility with charge storage nodes. This model is purely functional, i. e. it consists only of closed-form equations. Yet, it is also able to model classic MOSFETs like FinFETs [41] and stacked nanowire devices as found in [11] as special cases. I use this to evaluate the precision of this model against reliable known test data for the CMOS device from [41].

`prism-gen model` The transistor model itself is directly integrated in `prism-gen`. No additional discretisation is necessary, but the model directly works on the equations shown above. To satisfy operational demands when working with PRISM, I use a polarised model of the transistor that establishes $V_{\text{l}\to\text{r}} \geq 0\,\text{V}$ such

that the current $I_{\mathrm{l}\to\mathrm{r}} \geq 0\,\mathrm{A}$ flows out of the *right* contact. This reduces the number of cases for computing the directional currents and makes currents of p- and n-conducting transistors easier to compare with each other during early device characterisation. Obviously, the actual polarity is taken into account when connecting transistors to charge storage nodes.

A single transistor gate is described as:

```
(gate-def identifier [[ :threshold threshold-voltage |
                        :load gate capacitance |
                        :threshold-slope-n ss-nmos |
                        :threshold-slope-p ss-pmos |
                        :p-threshold-n corrected-Ith-nmos |
                        :p-threshold-p corrected-Ith-pmos |
                        :leakage leakage-current ]]*)
```

Example:

```
;; define a single transistor gate
;; currents and capacitances rescaled to μA, μF
(constant Vth double 0.4)
(gate-def sb-gate :threshold Vth
                  :p-threshold-p 9e-3 :p-threshold-n 2e-3
                  :threshold-slope-p
                  (pow (/ 0.4 9.5e-6) (1/ Vth))
                  :threshold-slope-n
                  (pow (/ 0.12 9.8e-6) (1/ Vth))
                  :load 40.0e-12)
```

A transistor gate definition `gate-def` describes a single manifestation of a transistor gate. This means it captures the characteristics of a gate of a particular technology node with considering all physical and geometrical parameters (as far as they influence electrical performance). Generally, arguments that describe currents are usually named `p-*`, because they do not necessarily literally describe currents but get adjusted by `prism-gen` to fit the charge transport model. The same goes for capacitances, which are usually called *loads* and may start with `h-*`, symbolising the load factor H. The reason is that loads describe equivalent capacitance values where the capacitance always connects the signal path to ground. In the current iteration, all series capacitances are converted into a resistance/current and a capacitance towards ground.

Incidentally, due to how the transistor model and the charge storage nodes are modelled, the currents and capacitances could be re-scaled to 1×10^{-6} to improve numerical stability. This shift was applied throughout all experiments and circuits that were created in this thesis and care must be taken to not mix models with differently scaled currents and capacitances. The example above shows values that are consistent with a Schottky gate of the germanium nanowire transistor from Figure 2.6 on page 26. These gate definitions are, then, used to

define transistor types:

```
(rfet-def identifier [[ :gates ({gate-definition}*) |
                        :beta-n channel-resistance-nmos |
                        :beta-p channel-resistance-pmos |
                        :h-channel-l channel-cap-left |
                        :h-channel-r channel-cap-right ]]*)
```

Example:

```
(constant Cchan double 40.0e-12)
;; create an RFET with 2 Schottky barrier gates and a MIGFET with 3 gates
(rfet-def 2gfet :gates (sb-gate sb-gate)
                :beta-n (* 2 (/ 13.94 (pow (- V_MAX Vth) 2)))
                :beta-p (* 2 (/ 11.65 (pow (- V_MAX Vth) 2)))
                :h-channel-l Cchan
                :h-channel-r Cchan)
(rfet-def 3gfet :gates (sb-gate md-gate sb-gate) ... )
```

Independent of a particular gate, the model needs to know the channel resistance δ^{POL} (in `prism-gen` still referred to as `beta-`*`pol`*) and the parasitic capacitances that the transistor channel contributes on either contact side. The transistor gates that are applied to these definitions in the `:gates` argument are ordered from left to right. Thus, the left Schottky barrier gate could have different electrical characteristics, when the physical design of the transistor is asymmetric. MIGFETs are created by naming more gates in the `:gates` argument. Inner gates, that do not reside over a Schottky barrier usually have substantially different electrical characteristics, which is why the example above refers to a separate (not otherwise shown) `md-gate` definition for the inner gate. As the examples show, the designer can use literal values, define constants and re-use them in definitions or describe parameters as formulae directly.

A complete circuit definition looks as follows:

```
(net 3majority (:nodes (A inv_A B inv_B
                        C inv_C out SUPPLY GROUND))
  ;; load library of transistors into lib and create 3 of them
  (load "transistor-def.net" lib)
  (rfet-node mgfet_R0 :type lib/3gfet)
  (rfet-node mgfet_R1 :type lib/3gfet)
  (rfet-node mgfet_R3 :type lib/3gfet)
  ;; create output charge storage node and connect all inputs/outputs
  (cap-net-node net_out :nodes (out mgfet_R0_r
                                mgfet_R1_r mgfet_R2_r))
  (connect A     mgfet_R0_l)
  (connect inv_A mgfet_R0_lg mgfet_R0_rg mgfet_R1_m1)
  (connect B     mgfet_R2_l)
```

```
(connect inv_B mgfet_R2_lg mgfet_R2_rg mgfet_R0_m1)
(connect C     mgfet_R1_l)
(connect inv_C mgfet_R1_lg mgfet_R1_rg mgfet_R2_m1))
```

The circuit, a 3-input majority logic gate, which is planned to be integrated into a larger experiment, is defined with an interface in the first two lines of the example. This interface consists of direct and inverted signals for the three inputs A, B and C, as well as the output signal `out` and the reference voltages. The example goes on to load another `prism-gen` source file, which contains transistor definitions like in earlier examples. They are immediately used to instantiate the three needed transistors. These instances provide a connection interface which is related to their instance names. Thus, the transistor `mgfet_R0`, which is created as a 3-gate MIGFET, provides the contacts `mgfet_R0_l`, `*_r` etc., according to the naming scheme shown in Figure 4.8 on page 77. The last block of code creates the charge storage node that connects all transistor drain contacts to form the circuit output and connects all charge transport node endpoints to a respective charge storage node.

Closer inspection of the example suggests that, apparently, the reference voltages V_{DD} and V_{SS} are not used in this circuit implementation. Actually, though, their use is implied in the instantiation of the charge storage node `net_out`. The `cap-net-node` construct features a `:references` argument (cf. page 75) which, by default, establishes `SUPPLY` and `GROUND` as the reference voltage for a particular node. Mentioning the references in the circuit interface ensures that the `cap-net-node` construct picks up whatever references are supplied when this circuit is loaded into a surrounding experiment description.

Example Technology Germanium Nanowire Transistor

Throughout this work, I use the following transistor model of the germanium nanowire technology shown in Figure 2.6 on page 26 whose data stems from a device shown in [56]. As a nanowire technology, it features a channel length of 48 nm with gate lengths (along the axis of the wire) of 24 nm. This leaves a gap between the two gates of 24 nm, because the gates overlap the channel by 50 % (cf. Figure 2.2,

1. 21).

The `prism-gen` model extends the device to a MIGFET by adding hypothetical inner control gates whose characteristics were drawn from additional TCAD simulations. Each inner control gate extends the channel length by 40 nm. I assumed $C_c = 40\,\mathrm{aF}$ for all transistor contacts, and a nominal supply voltage $V_{\mathrm{DD}} = 1.2\,\mathrm{V}$.

Figure 4.9 shows the I–V diagram of the `prism-gen` model as a heat map. The transistor, with its two gates lg and rg, spans the heat maps in their range between $V_g = 0\,\mathrm{V} \ldots 1.2\,\mathrm{V}$, the colour displays the drain current I_{D} flowing through the device, which is connected to $V_{\mathrm{l}} = 1.2\,\mathrm{V}$ and $V_{\mathrm{r}} = 0.0\,\mathrm{V}$. To highlight the

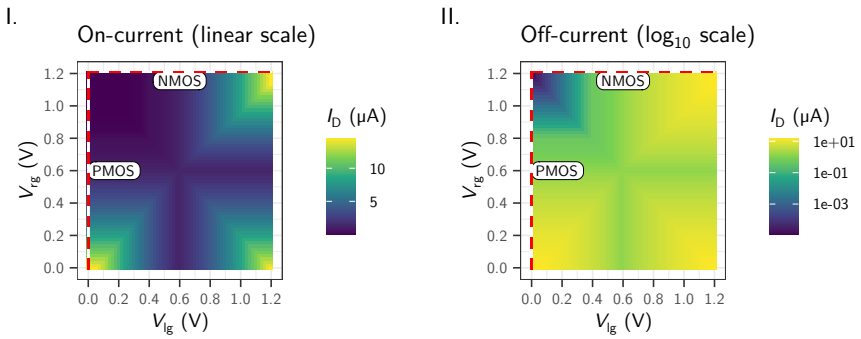


Figure 4.9: PRISM model of a single germanium nanowire RFET. 24 nm feature size, comparable to Figure 2.6 on page 26.

on-behaviour (i. e. large magnitudes), the left heat map uses a linear scale for showing the drain current I_D, whereas the right heat map uses a logarithmic scale to emphasise the off-behaviour (i. e. small magnitudes). The dashed lines mark the zones in which the device behaves like a conventional transistor subdividing the graphs each into the three normal operating zones between the off-state and the two on-states. The bottom-right corner shows the ambipolar behaviour of the device which is the result of taking the maximum of the possible currents, as described in Equation 4.4. This is an estimate made due to a lack of experimental data and insight into how exactly the device behaves in the ambipolar region.

Recalling Figure 2.6 (p. 26), the TCAD data shows that I_D in the full ambipolar case (bottom right corner) is higher than either full on-current, but it is also not the sum of both on-currents but less. This simulation result remains unexplained as of this writing, and due to lack of experimental data with a real device, it could neither be proved nor disproved. Figure 4.10 overlays the TCAD data with the PRISM data to show four error plots. Due to the limited simulation data, the x axis only shows five distinct data points. The top half displays the absolute error Δ_{abs} in micro Ampere, again, on a linear (left) an a logarithmic scale (right). In absolute terms, the NMOS and PMOS curves look as expected. The three corners are easiest to describe in the `prism-gen` model, which is why they are a good match with the TCAD data. The transients down the PMOS curve differ from each other for $V_{lg} = 0\,\mathrm{V}$. This can be explained by the fact that devices at this scale do not entirely behave according to the MOSFET equations. Most prominently, there is a large deviation in the ambipolar regime in which the device steers the channel wide open immediately after crossing the PMOS threshold voltage at $V_{rg} = 0.8\,\mathrm{V}, V_{lg} = 1.2\,\mathrm{V}$. A similar, while less pronounced, excitation can be seen along the $V_{rg} = 0\,\mathrm{V}$ axis.

Sub figure β shows that the on-currents are not exact matches although they

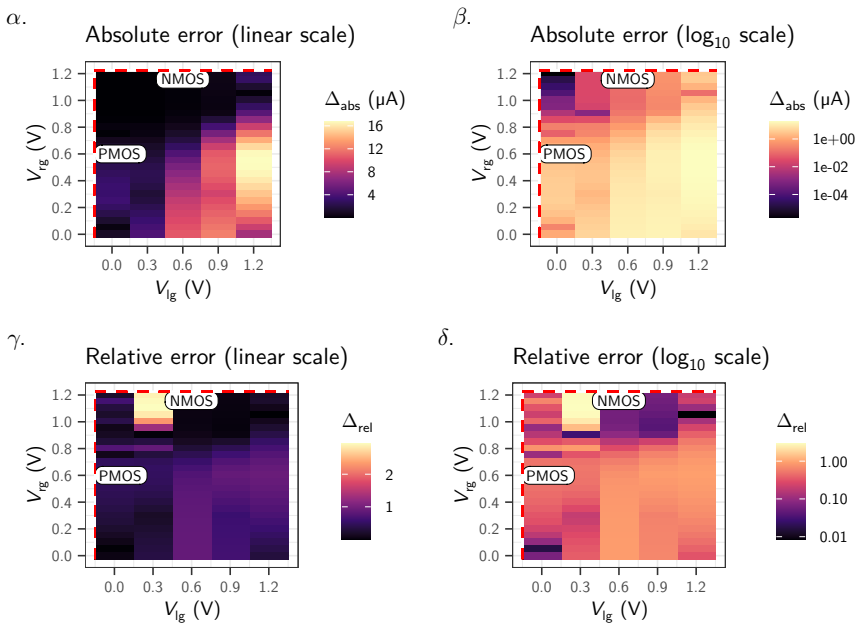
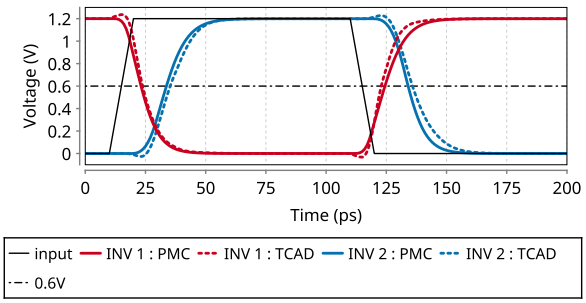


Figure 4.10: Difference between germanium nanowire TCAD simulation and the PRISM model.

can be determined relatively easily from experimental or FEM data. The reason is, that at those points the device barely enters saturation mode. I have no TCAD data for an overstressed device that is used outside its nominal operating regime, which would deliver relevant data to fit the curves of the MOSFET equations. My assumption, therefore, was to use conservative numbers which make the `prism-gen` model perform a bit weaker than the real device.

The lower half of Figure 4.10 shows the error in relation to the current's magnitude Δ_{rel}. What immediately catches the eye is the strong relative error of about 3 times around the threshold voltage for the NMOS behaviour. Oddly, the same deviation is missing for the PMOS behaviour. To explain this error, we must look at how the TCAD simulation arrived at its results. The TCAD model is a geometry- and material-based model, which incorporates properties like oxide thicknesses and crystal growth patterns. This underlines, that at those scales, devices are not free geometric forms but must adhere to a number of geometric constraints, which in turn determine how and at what magnitude their electrical properties can be influenced. One important electrical property was that NMOS and PMOS on-currents match as closely as possible. This is achieved by exciting mechanical stress on the nanowire via its all-around Schottky-barrier gates. The



Time (ps)	INV 1 PMC	INV 1 TCAD
$V_{out} = 0.6\,V$	23	24
	124	123
Time (ps)	INV 2	
	34	36
	134	136

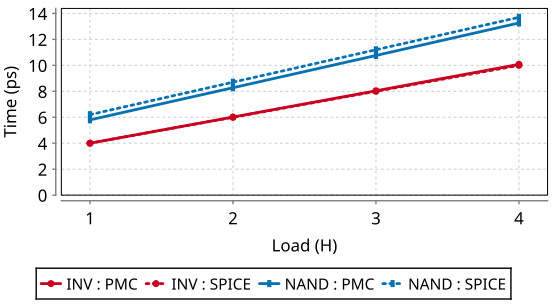
Figure 4.11: Time response of two buffers to an artificial input signal with a slope of $10\,ps \times {V_{DD}}^{-1}$. The solid curves show the `prism-gen` model, the dashed curves the TCAD model. $H = 1$ ($C_{out} = 80\,aF$)

stress influences electron and hole transport differently. It looks like this also slightly shifts forward the NMOS threshold voltage, such that the device starts conducting early. The currents around the threshold are very small, which is why this results in a high relative error that goes unnoticed on the absolute scale. Although the relative error goes up to 100 % in the intermediate states between the four corners, it must be noted that the current in theses regions is one to two orders of magnitude smaller than the on-currents. The graphs also show that the relative errors remains consistent across the device states, thus showing that the `prism-gen` model is able to capture the relevant properties of the device.

As a last remark, I want to note that the TCAD model of the germanium nanowire transistor which acts as the baseline of all qualitative and quantitative comparisons is at technology readiness level (TRL) 2. Working transistors have just been shown but neither working circuits nor circuit integration has taken place using this technology. This also shows, how early my modelling approach can be used to explore the logic circuit design space that opens up with a radically different device.

Circuit delay quantification error The question becomes: How large is the modelling error, and how does it influence circuit delay quantification? In my work on quantitative characterisation of reconfigurable logic gates [47], I showed that circuit delay quantification is close to the TCAD simulation.

Figure 4.11 shows the main result of quantifying the circuit delay error. The experiment was conducted as an FEM simulation using the TCAD model resulting in the dashed curves and using the `prism-gen` model. In both cases, the circuit under test consists of a buffer (i. e. two inverters in series) that drives a load equivalent to one inverter. The red curves show the first inverter that is fed with an artificial input signal with a slope of $10\,ps \times {V_{DD}}^{-1}$ and sharp corners. These excite a relatively sharp transient overshoot in the TCAD model as a result of high-frequency-sensitive negative virtual capacitances that are not modelled in



$$
\begin{aligned}
I_{\text{on}} &= 109\,\mu\text{A} \\
I_{\text{D0}} &= 0.8\,\mu\text{A} \\
V_{\text{DD}} &= 1.0\,\text{V} \\
V_{\text{th}} &= 0.3\,\text{V} \\
C_{\text{gate}} &\approx 127\,\text{aF} \\
C_{\text{chan}} &\approx 127\,\text{aF} \\
1\,\text{H} = 2 \times C_{\text{gate}} &\approx 254\,\text{aF}
\end{aligned}
$$

Figure 4.12: Comparison of circuit delays in relation to output load H. The red curves show perfect alignment after fitting the `prism-gen` model to SPICE for $H = 1$. The blue curves show a good alignment and perfect output load scaling invariance in the control experiment with a 2-NAND gate.

either the `prism-gen` model or the MOSFET equations. The table on the right presents the delay values in pico seconds at the points where the inverter outputs cross $^1/_2 V_{\text{DD}}$. It shows that the differences are stable at 1 ps to 2 ps through the whole buffer.

The germanium FET TCAD models are very early designs. So, it is important to compare my modelling approach also to a proven technology, for which I chose the CMOS 32 nm process from [41]. In this comparison, an inverter implemented in the CMOS `prism-gen` model is fitted to a reference inverter implemented in the pre-existing SPICE model, see Figure 4.12. It shows the input-output delay the circuits under test over an increasing output load $H = 1 \ldots 4$ inverter equivalents. The two red curves, which show the aligned inverters not only match for $H = 1$ but also scale correctly to higher loads. Thus, the `prism-gen` model can be expected to work in complex circuit designs just as well as in this micro-benchmark. The blue curves show the delays for a two-input NAND implementation employing the previously aligned device model. Although the results differ slightly depending on the model, the slopes match exactly with 2.5 ps/H. So, the modelling error remains invariant to scaling and thus to the surrounding context in which the device model is being used.

4.3.4 Queries for Quantitative Analysis

Apart from the network model and the efficient description of logic gates, the quantitative analysis needs queries that answer interesting questions or quantify relevant properties. Their use should enable circuit designers to gain insights into the trade-offs of different implementations of a single switching function. I consider, delay, power dissipation and energy consumption to be the main physical properties of interest in this regard. Implementation area, another critical measure for circuit designers, which is a function of the number of transistors and

the complexity of their interconnect, can be answered without employing model checking. Ideally, the queries implementing the quantification of these physical properties are independent of a particular circuit implementation, and I show that their dependence can be reduced to the number of inputs and the model quantisation parameters V_{scale} and T_{scale}.

Advantage of model checking One advantage of model checking over simulation techniques is that it can query the results directly from the model without the need for the experimenter to present the model checker with (or even know of) the critical input patterns that will excite the circuit to deliver the result. The analysis in the next chapter shows that relevant results sometimes hide in input patterns that are usually excluded from simulations to reduce the number of stimuli that have to be tested. This also means, that model checking delivers the offending input patterns to extremal queries as witnesses, in addition to the result. Not only does the experimenter not need to know about the relevant patterns beforehand but they are also a direct result of the quantitative analysis. These features enable a thorough investigation over a wide range of circuit implementations as a push-button technique with no manual intervention.

For proper operation, quantitative analysis relies on the exploration of a *program graph*, that is the product of two graphs. The first is the model, which consists of the circuit network model and the input automaton that excites its inputs, while the second is the query. In the model, the input automaton must be correctly interlocked with the network model to produce a meaningful graph. It is easy to see, that the input automaton should only advance once the circuit network model has itself advanced to a certain state, which I call the *stable state*. This work focuses on the exploration of single-stage logic gates, and due to the typical structure of MIGFET circuits, that favour a transistor path lengths of 1, there are no intermediate charge storage nodes between the power supply nodes and the output node. Additionally, this work focuses solely on complementary static logic gates, which means they are truly memory-less (i. e. have no feedback paths). So, it is sufficient to observe the outer connections of a circuit to observe whether it is stable or still transitioning. Let the set of charge storage nodes that are either controlled by an input or constitute the output, $I \subseteq S$, be called the *interface* of a circuit. The circuit has stabilised iff. all interface nodes have stabilised (cf. Equation 4.3):

$$stable \stackrel{\text{def}}{=} \bigwedge_{s \in I} stable_s$$

This criterion is used by the input automaton to control its advance. As described in previous sections, the stable criterion is accessible to the designer through the `:stable` argument, which allows the addition of intermediate charge storage nodes for more complex circuit designs.

Circuit delay In the circuit design community, there is no fixed standard on how to measure circuit delay and tool suppliers like Cadence and Synopsys come with their own defaults. Exact information on the start and the ending of a switching transient are also hard to obtain for these tools, which is why the community uses various rules of thumb for the definition of those two points which define the measure, e. g. a 10-90 delay starts the measurement when the input crosses 10 % of its full swing and stops when the output crosses 90 % of its transient curve. Additionally, there is no single true measurement of circuit delay, because it may depend on the context in which the circuit shall perform which determines the relevance of the shape and steepness of input and output transients. And their relevance determines whether much of them should be counted into the delay or may actually mask the relevant circuit performance details that the designer tries to extract. In this work, I use those states in the program graph as the starting point of the quantification at which the input automaton advances, triggering a change in the inputs. At this state, the input's charge storage node becomes unstable but still assumes its equilibrial voltage level. The set of start reference states *switch* that trigger a change in inputs $\widehat{I} \subset I$ is defined as:

$$\begin{aligned} \mathit{switch} &\stackrel{\text{def}}{=} \bigvee_{i \in \widehat{I}} \mathit{switch}_i \\ \mathit{switch}_i &\stackrel{\text{def}}{=} \neg \mathit{stable}_i \wedge (D_i = D_{\mathrm{SS}} \vee D_i = D_{\mathrm{DD}}) \end{aligned}$$

The ending reference point relates to the output and is set to be the crossing of the half swing. In certain designs, though, the output may cross $1/2 V_{\mathrm{DD}}$ multiple times before finally stabilising at it intended value, a phenomenon known as glitching or output hazard. For the delay quantification, only the last crossing shall be used as a reference, which, due to perfect knowledge, can be achieved with model checking. The set of ending reference states cross^{ω} for a number of outputs $O \subset I$ is defined as:

$$\begin{aligned} \mathit{cross}^{\omega} &\stackrel{\text{def}}{=} \bigvee_{o \in O} \mathit{cross}_o \wedge \bigwedge_{o \in O} \begin{array}{l} o \text{ has crossed after } \mathit{switch} \text{ and won't cross} \\ \text{again on all paths of program graph} \end{array} \\ \mathit{cross}_o &\stackrel{\text{def}}{=} \mathit{falling}_o \vee \mathit{rising}_o \\ \mathit{falling}_o &\stackrel{\text{def}}{=} (V_o > 1/2 V_{\mathrm{DD}}) \wedge V_o \leq 1/2 V_{\mathrm{DD}} \text{ on all paths of program graph} \\ \mathit{rising}_o &\stackrel{\text{def}}{=} (V_o < 1/2 V_{\mathrm{DD}}) \wedge V_o \geq 1/2 V_{\mathrm{DD}} \text{ on all paths of program graph} \end{aligned}$$

The first line ensures that there are no more crossings of a particular output after the current *switch* occurred. It is specified in temporal logic, and its exact definition is not beneficial for the reader and would require extensive introduction into the specifics of the temporal logic language (see [3] Chap. 6). Thus, it is left out of this document and may be referred to via the queries source file attached to this work. The definitions for *falling* and *rising* ensure that the output is on one side of the reference point and will cross to the other side.

For a correct delay calculation, the ending reference must relate to the same *switch* as the start reference. Thus, we may only consider those states where an input switch causes an observable effect at the output.

$$\mathit{effect} \stackrel{\text{def}}{=} \bigvee_{o \in O} \mathit{effect}_o$$

$\mathit{effect}_o \stackrel{\text{def}}{=}$ *switch* happened and no further *switch* happens until output o crossed

Using these effective selectors, the worst-case delay $\Delta t_J^{\max}$ after switching any input $J \subseteq I$ can be described as: the maximum over all expectancy values, that count discrete time steps from *switch* to cross^{ω}, over state $s \in J$ that perform a *switch* that also has an *effect* on the output.

A similar metric is defined for the average expected delay Δt^{avg} in which the probability that a particular *switch* that excites an *effect* occurs is taken into account. Also, the reader can take from above definitions, that it is straightforward to to match other delay definitions by adapting the start and ending references switch_i and cross_o.

Power dissipation and energy Due to the design of the network model, I can read the momentary power consumption directly from the model. For each charge transport node T in the model, its current voltage difference V_t between two connected charge storage nodes and the partial current through the node I_t is always known. Thus, momentary dynamic power dissipation is defined as:

$$P \stackrel{\text{def}}{=} \sum_{t \in T} V_t \times I_t$$

The maximum dynamic power dissipation is, thus, computed over all states in which the circuit is switching. These switching events, though, need not necessarily be affecting the output.

The same is done for computing energy consumption per operation. Due to the discrete model, the integral equals the sum of momentary power values over all model states that occur between the start and the end references. Each time step has the same length, which is T_{scale}. Thus, rescaling the sum with T_{scale} directly yields the energy consumption for the whole time interval.

4.4 Circuit Variant Generation

The last part of my approach to a comprehensive design space exploration of reconfigurable logic gates is the generation of circuit variants that can be quantified. As explained in Section 2.1.2, the only logic gates that can have multiple implementations as complementary static logic circuits, are reconfigurable circuits, which must be implementing a (partially) self-dual Boolean function. Thus, the

circuit variant generation concentrates on unfolding design space in terms of these functions.

4.4.1 Function Expansion

The expansion algorithm targets single-stage implementations of static logic circuits of the size of standard cells. Its input is a self-dual Boolean formula specifying the switching function. The output is set of implementations as `prism-gen` circuit descriptions. Any (partially) reconfigurable circuit uses three sets of transistors, those that are statically configured to PMOS and NMOS and reconfigurable transistors. The main goal of the algorithm is to *minimise* these three sets of transistors. This means that certain special-purpose implementations that favour a balance between P- and N-branches in the number of transistors will be excluded from the results. Examples of these balanced implementations can be found in [2]. The algorithm was largely developed by Steffen Märcker, who is my co-author of [47], where we first published its application. It was first published in [34]. The foundation of the algorithm is the well-known Quine-McCluskey Algorithm [35], whose idea is to compute a set M of minimal product terms, called *prime implicants*, that, applied in a disjunction, compute the equivalent to a Boolean function given by a set F of product terms, i. e. $M \iff F$. As an extension to the Quine-McCluskey algorithm, prime implicants may imply additional minterms from a set of *optional terms* O, such that, $F \Rightarrow M$ and $M \Rightarrow F \cup O$. For an n-ary Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}$ there be:

$$P_f \stackrel{\text{def}}{=} \{(\neg b_1 \wedge ... \wedge \neg b_n) \mid (b_1 \wedge ... \wedge b_n) \text{ is minterm in DNF of } f\}$$
$$N_f \stackrel{\text{def}}{=} \{(b_1 \wedge ... \wedge b_n) \mid (b_1 \vee ... \vee b_n) \text{ is maxterm in DNF of } f\}$$

where $\neg b$ describes the negated literal b, eliminating double negation. A set T of terms is *compact* if there is no strict subset T' equal to T, i. e. $\nexists T', T' \subset T . T' \iff T$. For a Boolean function f, the algorithm works as follows:

1. Enumerate the tuples of minimal term sets P, N and R that make up the circuit transistors:
   (a) Compute set of dual terms $R_f = P_f \cap N_f$
   (b) Compute set of prime implicants R'_f of R_f using Quine-McCluskey Algorithm
   (c) For each compact set $R \subseteq R'_f$ and optional terms $O \subseteq R_f$ such that $O \iff R$:
      i. Compute set of prime implicants P'_f of $P_f O$ using Quine-McCluskey Algorithm including optional terms O
      ii. Compute set of prime implicants N'_f of $N_f O$ using Quine-McCluskey Algorithm including optional terms O

iii. Yield tuples (P, N, R) for all compact sets $P \subseteq P'_f$ and $N \subseteq N'_f$ such that $P \iff P'_f$ and $N \iff N'_f$ that are unique (not equivalent under variable renaming)

2. Enumerate reconfiguration mappings $r\colon R \to \{x_1, \ldots, x_n\}$ for each tuple (P, N, R):

   (a) Yield a *balanced* mapping $\tilde{r}$ such that all reconfiguration variables appear equally often

   (b) Yield *biased* mappings r_k for each variable x_k occurring in all terms in R that are unique (not equivalent to each other under renaming) such that $\forall t \in R.r_k : t \to x_k$

3. Yield a circuit for each tuple (P, N, R, r):

   (a) Instantiate a transistor for each term in P and N according to the "inner" reconfiguration mode (cf. Table 2.3)

   (b) For each reconfiguration mapping r and the reconfiguration modes "inner", "single" and "transmission":

      i. Instantiate a transistor for each term t in R according to reconfiguration mode and reconfiguring the transistor according to $r(t)$

As thoroughly described in Section 2.1.2, a reconfigurable function can be considered a higher order function in the reconfiguration variable that selects between two subordinate functions, and this is equally described by Shannon decomposition. Depending on application, if this selection between two sub-functions occurs sufficiently rarely, there may be a benefit to a biased reconfigurable circuit that puts a high load onto the single reconfiguration variable, as is done in Step 2.b of the algorithm. So, a design space opens up between balanced implementations that try to equally distribute the load across all inputs and biased implementations that may yield maximum performance for some of its inputs. To generate all reconfiguration variants, Step 2 has to be repeated for each input variable and the Quine-McCluskey Algorithm is modified to prevent it from removing the respective variable of interest in the computation of R'_f in Step 1.

Each function f will contain one tuple $(P, N, \emptyset)$, which is a *fully static* implementation. For each function f that is self-dual, the algorithm will yield tuples $(\emptyset, \emptyset, R)$, which are *fully reconfigurable* implementations. All other tuples (P, N, R) will yield *semi-static* implementations. The algorithm enumerates every circuit, it creates, in the scheme f, i, μ, m where f is the function name, i is the running index of the tuple (P, N, R, r) in the list of generated tuples, μ mapping that was used, i. e. $\mu = \{\tilde{r}, r_1, ..., r_k\}$ for a k-ary function, and m is the reconfiguration mode that was used in Step 3.b. For example a 3-XOR circuit might yield an implementation called: 3-XOR,2,x_0,in.

Since the algorithm has to consider each subset of dual terms in Step 1, it has to consider at most $2^{2^{n-1}}$ variant, multiple minimal representations of each term

not accounted for. This makes the worst-case run time double-exponential in the number of variables. Nevertheless, the prototypical implementation used in this work showed acceptable performance for switching functions up to five inputs. A study of several Boolean functions their design space and their quantitative analysis is shown in the next chapter.

Chapter 5

Quantitative Analysis of Standard Cells

We have seen in the previous chapters how polarity-controllable transistors lend themselves naturally to build certain kinds of reconfigurable circuits from them. These circuits implement self-dual Boolean functions, which, by their nature, match perfectly with the way FETs are operating, such that each pair of dual terms can be implemented with the same set of polarity-controllable transistors. Chapter 2 describes the process in detail and concludes that, for transistors which use electrostatic channel polarisation (as opposed to chemical polarisation with dopants) through additional gates, this opens a new design space along various axes. The previous chapter explained the mechanics of the design space exploration in full and the evaluations in this chapter will make use of all the device and circuit models provided by it, especially the 24 nm germanium nanowire transistors.

The first axis of the design space exploration conducted in this chapter stems from the fact that multiple sets of "minimal" terms can implement a reconfigurable function. This axis is guided by the cover of reconfigurable terms over the whole set of terms that implement the target function. There is always at least one minimal set of terms which is not covered by any reconfigurable terms (thus, representing the static variant). The function might be covered completely by multiple sets of reconfigurable terms, and there may be sets of terms that partially cover the target function. This axis is called the *topological axis* of the DSE.

The second axis describes that transistor polarity control operates in the signal domain, which means that circuit input signals can be naturally used as reconfiguration inputs. One property of mutually exclusive terms is, that they can be reconfigured in any input, because each input in one term has a dual in the other term. This means that every self-dual Boolean function can be reconfigured via one or more of its input signals. While being functionally equivalent,

these circuit implementation variants will have varying performance characteristics. While self-duality usually imposes the need for having the inputs available as direct and negated signals, some Boolean functions feature variants that can save on one of the input inverters. This gives them dramatic energy and power savings compared to faster, more balanced variants (cf. Figure 5.6, page 107). In contrast, reconfiguring a circuit in exactly one input may reduce the load on the other inputs. In some applications, this dramatically decreases the critical path delay, when the slow input can be routed to a non-critical path. The 3-MIN and 3-XOR logic gates in Figure 3.3 show, how this can be used to an advantage in a larger circuit design. This axis realises the *reconfiguration type* and expresses that reconfiguration is either achieved by using one particular input, or it may be balanced across multiple inputs.

The third axis of the design space of reconfigurable circuits lies in the input connections to each of the transistors. The nanowire transistors used in this work are semiconductor-metal heterostructures that erect Schottky barriers at either end of their channel. Thus, their two outermost transistor gates enclose the Schottky barriers (cf. Figure 2.2) and need to perform more electric work than the inner transistor gates that are located around the semiconducting part of the wire. While the polarity control gate is always located at the acting drain side of the device, the other gates can be freely assigned to any input, which results in the transistor *reconfiguration modes* described in Table 2.3 and that constitute this axis.

The DSE that is conducted in this chapter orients itself largely along these three axes. A particular point in all three axes describes a single *circuit variant*, sometimes called an *implementation* for better distinction. I show an analysis of the smallest possible self-dual Boolean function, 3-input minority, followed by the 3-input XOR function. The results will show that there is no simple answer to the benefits and drawbacks of each implementation. While some implementations clearly outperform others, there are surprising benefits of variants that seem not to perform well but do under certain constraints.

In this thesis and especially in this chapter, the term *static circuit* bears a different meaning than usual for describing static CMOS logic gates. A circuit or a part of a circuit is called *static* if it is not reconfigurable, i. e. the polarity-control gates of affected transistors are connected to a static signal. A circuit is called *semi-static* if it consists of static and reconfigurable parts.

All circuits constructed in this chapter are always static logic gates in the sense that their outputs assume a stable fixed value eventually for stable fixed input values as opposed to dynamic logic gates, which need timely set and read cycles to produce correct results. Also, all variants created by the DSE algorithm are always complementary logic gates, regardless whether they are reconfigurable, semi-static or static.

5.1 Analysis of 3-Input Minority Logic Gate

Chapter 2 shows that the 3-input minority function is one of the simplest logic functions that are self-dual and, thus, have reconfigurable variants. In this section, I show the results from the circuit DSE and explain the particulars of the most interesting variants. This is followed by multiple experiments on the whole family of circuit variants to carve out the design space that is spanned up by the family. The structure of its circuits requires the use of inverted input signals, which can likely be shared with other logic gates in large circuit designs. So, in addition to showing their performance results in the time, power and energy domain, I also show their susceptibility to input inverter load and output load in the worst case.

5.1.1 Circuit Variants

The automatic exploration of the 3-input minority function yielded seven distinct circuit topologies enumerated 1–7. In this evaluation, the biggest topology number always describes the static implementation, regardless of the particular circuit. Each of these topologies can implement one or more reconfiguration types and modes. For the 3-MIN, the only topology with two reconfiguration types is number 3. Incidentally, the 3-MIN function does not support the transmission gate reconfiguration mode, because none of its topologies provides a combination of transistors and inputs that exclude short-circuits under all input conditions. So, the transmission gate reconfiguration mode is left out in this analysis. This reconfiguration mode is not to be confused with driving a transistor with an input signal, which is commonly referred to as transmission gate or pass gate logic. The transmission gate *reconfiguration mode* describes that the signals connected to the transistor gates, including the polarity-control gate, are *independent* of the input signal at the source contact, see Table 2.3 on page 33 for a detailed explanation. All topologies and variants have been first described and quantified in my joint work with Steffen Märcker in [47]. Before that, only a few variants have been known and crafted manually.

It would be tiring to present them all in full in this work, because after understanding the concept, there are strict rules to build each variant. Figure 5.1 presents an alternative form as stick diagrams. The sticks resemble the circuit topologies through their orientation, where horizontal lines stand for transistors that are reconfigured, top vertical lines describe the static pull-up network and lower vertical lines describe the static pull-down network connected to their respective power rails. It turns out to be sufficient to annotate the reconfigurable branches with their respective input signal to fully describe the reconfiguration types that these variants implement. Topologies are counted with numbers and their reconfiguration types are added to form principle variant names like 1,bl or 6,bl/A. Thus, while in Figure 2.10 I, the variant 2,A was drawn with vertical transistors showing a clear separation into what seems to be a pull-up and pull-down network, this was only done to achieve immediate recognition as a classical CMOS

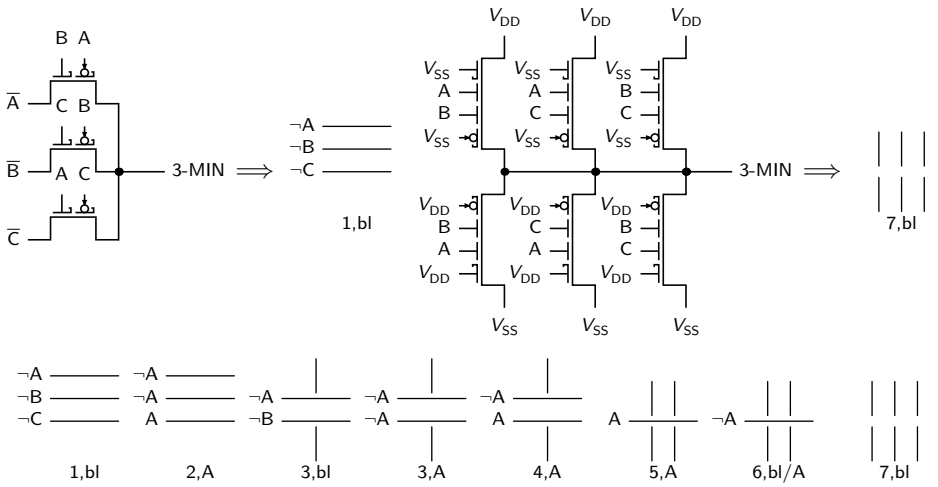


Figure 5.1: The eight principle variants of 3-MIN as stick diagrams. Horizontal lines depict reconfigurable branches denoting the source input. Thus, the variant 1,bl, top left, can be written as shown with three lines on top of each other. Vertical lines show the PMOS/NMOS branches. The fully static implementation, having no reconfigurable branches, needs six branches overall.

circuit. In the stick diagrams, variant 2,A can be seen as three horizontal sticks, all of them annotated with input A or its inverse. It still is a complementary design, but no longer static.

The reconfiguration modes, *single* and *inner* in case of the 3-MIN, are left out, as they do not contribute to the circuit's functionality. It must be noted, though, that static branches are always implemented in inner mode because of its performance benefits in a static configuration. This needs to be reconsidered, once leakage becomes a notable influence on the overall power budget that might outweigh the performance penalty of Schottky-barrier gates. Also, when the transistor type does not show the performance differences between inner and Schottky-barrier gates, the single modes may be beneficial for static branches, as well.

Regarding the 3-MIN function, variant 1,bl was found via this DSE and is also shown in full in the top-left corner of Figure 5.1. It is easy to see why this variant uses a balanced reconfiguration type; each input signal reconfigures exactly one of the three transistors, whose connection pattern visually resembles the minimized Boolean function of 3-MIN:

$$f_{\text{3-MIN}} = \left(\overline{\mathsf{A}} \wedge \overline{\mathsf{B}}\right) \vee \left(\overline{\mathsf{B}} \wedge \overline{\mathsf{C}}\right) \vee \left(\overline{\mathsf{A}} \wedge \overline{\mathsf{C}}\right)$$

Variant 2,A is the well-known implementation shown in [23] and Figure 2.10 I. It is also described as a NAND/NOR circuit because of its physical structure.

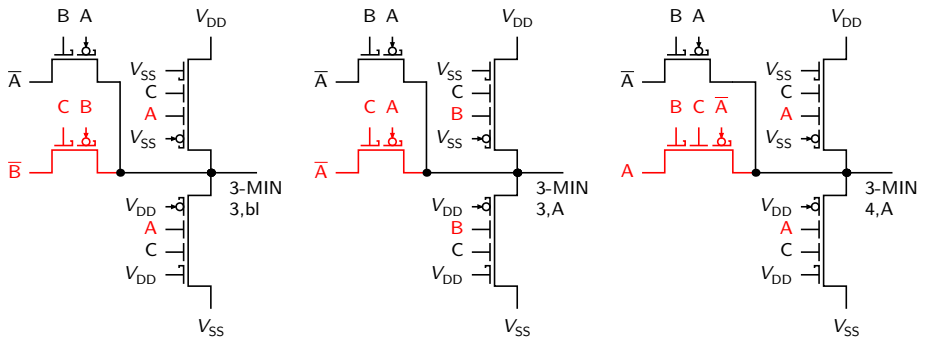


Figure 5.2: Topologies 3 and 4 of 3-MIN show the difference between a balanced reconfiguration and a reconfiguration that is restricted to a single input. Variant 3,A highlights how the complementary restricted reconfiguration in A makes it a different topology from 4,A, necessitating a change in the input connections to all four transistors.

Reconfiguration in a single signal, without loss of generality, A, forces two transistors to reconfigure on $\overline{A}$ and one transistor to reconfigure on A. As self-dual functions are invariant in the choice of their reconfiguration inputs, the variant notation 2,A means that a circuit designer can also use inputs B or C at their discretion. See also Table 2.2 for an example truth table and further explanation.

Variant 3,A is a similar implementation. It shows, that taking away a single reconfigurable branch, reducing from three to two, must be accompanied by adding two new static branches. Either static branch can only perform half the work of a reconfigurable branch, which also explains the relation of three reconfigurable branches in variant 1,bl to six static branches in 7,bl. Other than that, it keeps the complementary nature of the reconfigurable branches. While this assessment is correct in this narrow focus on the circuit structure, replacing two static transistors with a reconfigurable one is usually paid up with adding an inverter to produce the reconfiguration signal. This means, two transistors are actually replaced with three. It is easily seen that the break even (in numbers of transistors) occurs at replacing four static branches with two reconfigurable ones that are reconfigured in the same input. This is exactly what happens in variants 3,A and 4,A while variant 2,A even reduces the overall number of transistors. The effects on power dissipation and dynamic energy consumption are complex and are subject of the following pages.

Reconfiguration in a single signal does not always result in a circuit structure that uses complementary reconfigurable branches, though. Topology 4 employs a balanced reconfiguration type and a reconfiguration in $\overline{A}$ that is not complementary. Figure 5.2 highlights the subtle differences that distinguish topology 3 from 4. Although the transistor pattern looks exactly the same, switching from topology 3 to 4, all transistors need changed input connections, because the underlying Boolean terms to create either topology are different. Switching from variant 4,bl

to 4,A, however, only involves reconnecting the inputs to the highlighted transistor. This topology is the only one for the 3-MIN function that facilitates both reconfiguration types (balanced and restricted to a single input). The DSE algorithm generates those variants of either type that distribute load across the inputs most evenly.

What might go unnoticed at first glance is, that the variants shown in Figure 5.2 have quite different power signatures, as well. The variants 3,A and 4,A need only invert one of their input signals, A, while the balanced variant also needs the signal $\overline{\mathsf{C}}$. Additionally, topology 4 always puts predefined loads on its three input signals, because none of them directly drives the output, while topology 3, due to its complementary implementation, needs to employ one of its inputs as a driver.

Topologies 5 and 6 share the same similarities as topologies 3 and 4. Again, apart from having a single reconfigurable branch, they were constructed from different input terms. Also similar to topology 3, topology 5 must exclude the reconfiguration signal A in all static branches (cf. Fig. 5.2, left), while topology 6 can distribute the input signals more evenly, which is why the variant is called 6,bl/A. One could view topology 6 to be a closer relative to variant 4,bl, unfolding the highlighted transistor into two static branches controlled by signals A and C. As this leaves only signal A for reconfiguration, although it was constructed as a balanced reconfiguration type, this variant gets the double name 6,bl/A.

There is nothing of surprise in topology 7. It, too, reflects the minimized 3-MIN function, just that the complementary output values must be generated explicitly, doubling the amount of needed transistors. An implementation with single-gate transistors would make use of signal sharing between parallel paths; but due to the lower virtual series resistance of multiple-gate devices, it is almost always beneficial to just double the input signal to avoid signal paths through two transistors. Also, as mentioned earlier, all transistors use inner mode connections, i. e. the source-side Schottky barrier gate is connected to its respective source voltage and not used for an input signal. This makes the circuit even larger but, given the intended device technology, shows the best speed and energy characteristics.

5.1.2 Worst-Case Analysis

It is common practice to judge circuit performance based on its worst-case characteristics. This makes sense regarding the ubiquitous presence of sequential logic in large circuits. The first analysis is done on all circuit variants, i. e. each topology, reconfiguration type and mode. For the 3-MIN function, this amounts to 15 circuits.

The parameters of this worst-case analysis shall be defined as follows:

- Each circuit is loaded with a single inverter, $H = 1$.

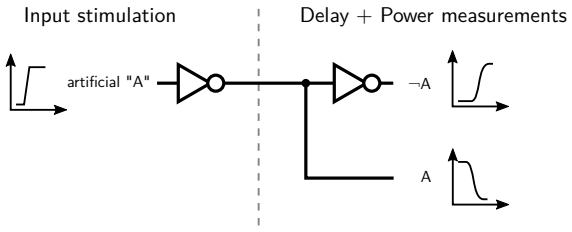


Figure 5.3: Input stimulation and conversion into analogue signals. Both inverters are modelled and use the transistor technology of the circuit under test.

- Modelling parameters are set to:

$$T_{\text{scale}} = 10^{-12} \qquad T_{\text{props}} = 10^{-13} \qquad V_{\text{scale}} = 10^{5}$$

- Each input is driven by an artificial ramp with a full-swing delay of 1 ps which is first converted to a physical input signal by an inverter of the same technology before connected to the circuit under test. Delay quantification starts at the outputs of these inverters (see also Figure 5.3).

- Each physical input signal X may be connected to a second inverter to produce the inverted signal $\overline{\mathsf{X}}$ when the inverted signal is needed. The inverter producing $\overline{\mathsf{X}}$ is *always* included in the delay calculation.

- The input automaton generates all combinations of full-swing signal transitions. The circuit under test is in equilibrium before a new input transition is explored.

- The quantified delay t_{max} is the worst-case delay.

- The quantified power dissipation P_{max} is the worst case, independently determined from t_{max}. Thus, P_{max} and t_{max} do not necessarily reflect the same switching event.

- The quantified dynamic energy consumption E_{max} reflects the maximum energy consumption for the (energetically) worst-case switching event. Static energy consumption is disregarded. E_{max} over t_{max} is quantified for two sub-cases:

  - Worst-case over all three inputs, $(E_{\text{max}}, t_{\text{max}})_{\mathsf{A,B,C}}$
  - Worst-case over two inputs excluding input A, which is the preferred reconfiguration input, $(E_{\text{max}}, t_{\text{max}})_{\mathsf{B,C}}$

- The three experiments $(P_{\text{max}}, t_{\text{max}})_{\mathsf{A,B,C}}$, $(E_{\text{max}}, t_{\text{max}})_{\mathsf{A,B,C}}$ and $(E_{\text{max}}, t_{\text{max}})_{\mathsf{B,C}}$ are performed for two sub-cases:

– The power dissipation / energy consumption for the input inverters producing $\overline{\mathsf{X}}$ *is* considered and affects the overall results, writing power as $P_{\max}^{\text{pwr}}$ and energy as $E_{\max}^{\text{pwr}}$.

– The power dissipation / energy consumption *is not* considered and the circuits are quantified in isolation, writing power as $P_{\max}^{\text{iso}}$ and energy as $E_{\max}^{\text{iso}}$.

This analysis results in six scatter plots shown in Figure 5.4 on page 103. They present all 15 variants as points, whose fill colour is determined by their topology; the darkest coloured points represent the variants with the most numerous reconfigurable branches. Each point is encircled with a coloured border, which represents the reconfiguration type. The shape of the points represents the reconfiguration mode, where the triangle shows single mode and the circle shows inner mode configuration. Thus, the static variant comes out as the bright yellow circle with a dark purple border. All Pareto-optimal circuits are connected by a black line. Each of the six plots is scaled exactly the same to allow visual comparison and to show where the results of each experiment fall "inside the box". The wide plots show $(P_{\max}, t_{\max})_{\mathsf{A,B,C}}$ and the left half-wide plots show $(E_{\max}, t_{\max})_{\mathsf{A,B,C}}$. Their delay values on the x-axis are exactly the same.

$P_{\max}$ and $E_{\max}$ for three inputs There is a lot to unpack from this experiment, so, let us focus on the wide plots α and δ, first. The plots show, that the reconfigurable 3-MIN variants can, indeed, work very power-efficient by sacrificing delay performance. In the isolated case (δ), the power savings become almost six-fold between variants 2,A,sg and 7,bl,in. Variants 1,bl,* are highly susceptible to whether inverters are counted towards the power budget or not. They are the only principle variants to employ three inverters, so the number of transistors ranges between 3 and 9 depending on whether inverters are counted or not. It is still worthwhile to look at the power budget in isolation, because the reconfigurable variants can expose their input inverters to the surrounding circuit. These signals are not available when using the static variant 7,bl,in and their power budget must be added on top in this case. Thus, the validity of the shown results highly depends on the context they are used in. A prospective process development kit will benefit from having both quantified models in its library. The models that include inverters would expose three inputs while the models without inverters would expose three to six inputs and could leave the optimisation to the technology mapper.

The half-wide energy consumption plots β and ζ on the left show, that for the worst case scenario, in which the inverters cannot be used elsewhere in the circuit, the static variant 7,bl,in is the absolute optimum; no circuit is faster or consumes less energy per operation. In isolation, however, principle variant 1,bl,* can perform their operation using less energy, despite being considerably slower.

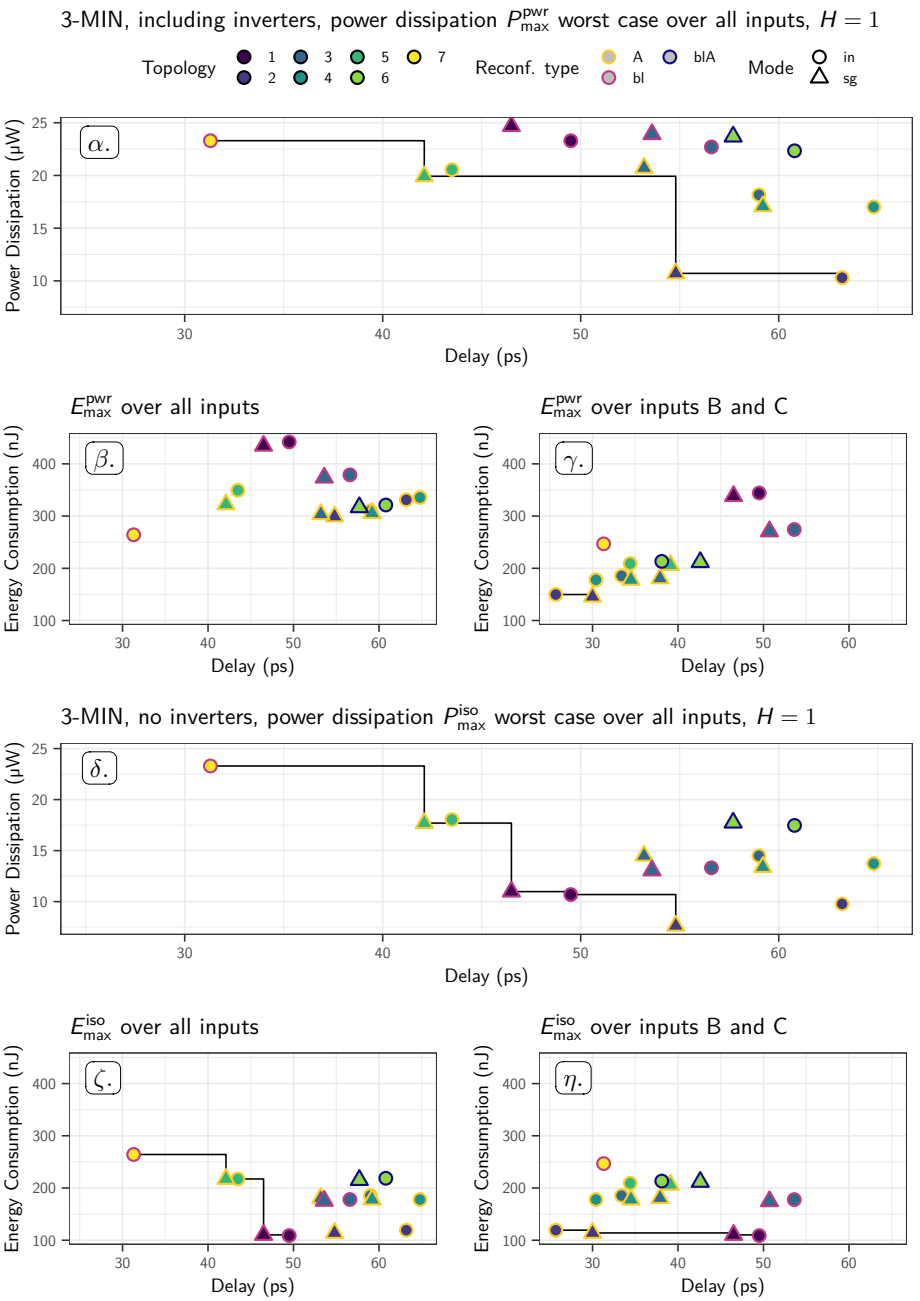


Figure 5.4: 3-MIN worst-case analysis. Results in top half consider inverters producing $\overline{A}$, $\overline{B}$, $\overline{C}$ to be part of the circuit, affecting power dissipation and energy consumption.

E_{max} without reconfiguration input The other half-wide plots γ and η on the right of Figure 5.4 consider only events that do not involve switching input signal A; they consider, however, both cases for A being fixed to 0 and 1. These cases can be viewed as showing the circuits' behaviour when operated as a reconfigurable NAND/NOR, opposed to the left plots which treat the circuit as 3-MIN. Reconfiguration happens orders of magnitude less often than signal changes during normal operation and the right half-wide plots show the extreme case, in which reconfiguration performance is disregarded. Their distributions look quite different from the plots on the left. In the upper-side plot, all circuits consume less energy on the right compared to the left. This is expected, because a) the input switching events are less extreme, involving at most two inputs and b) all variants need an inverter when reconfiguring in A and this is not counted (because A never switches). It is obvious that the static variant 7,bl,in cannot take advantage of this scenario. Some reconfigurable variants, however, not only gain in energy efficiency but also outperform the static variant in delay.

Transistor reconfiguration allows a considerable reduction of parallel paths. For the given circumstances, the static variant cannot make use of its available parallel paths, losing a lot of energy in short-circuit currents in a switching event without gaining much performance. It even costs performance compared to variants 2,A,in and 2,A,sg, which become the new Pareto-optimal variants. They completely benefit from the fact that they can perform the same sub-function (either NAND or NOR, depending on input A) with half the number of transistors. Variant 2,A,in cuts its delay down from 63 ps to 26 ps. Depending on the application, this may well be the faster variant *and* more energy efficient, i. e. when input A only switched in a completely different timing regime. This is similar to an FPGA switch box, where particular signals are always static during operation and only switch during reconfiguration, while others in the same switchbox must act on user signals.

In isolation, the results are even more pronounced. All reconfigurable variants outperform the static variant in energy consumption per operation. Under these conditions, principle variant 1,bl,*, which is reconfigurable in all inputs, reduces dynamic energy consumption to as low as 110 nJ, which is in the order of a single inverter.

Reconfigurability can be used to "unload" the inputs of the sub-functions and concentrate the load in the reconfiguration input. Variants 7,bl,in (yellow circle) and 1,bl,* (purple triangle/circle) demonstrate perfectly balanced circuits, in terms of input load distribution. Each of their inputs must perform equal work to make the circuit switch. Their energy and delay results are very similar to each other regardless of whether two or three inputs are allowed to switch. Now, by putting most of the effort on the reconfiguration input, we can create very fast and power-efficient variants. The most extreme cases are variants 2,A,in (grey/yellow circle) and 5,A,in (green/yellow circle). On the left in β and ζ, they are the variants with the worst delay performance (about 65 ps). On the right, though, in γ and η, they are the two variants that outperform the static implementation

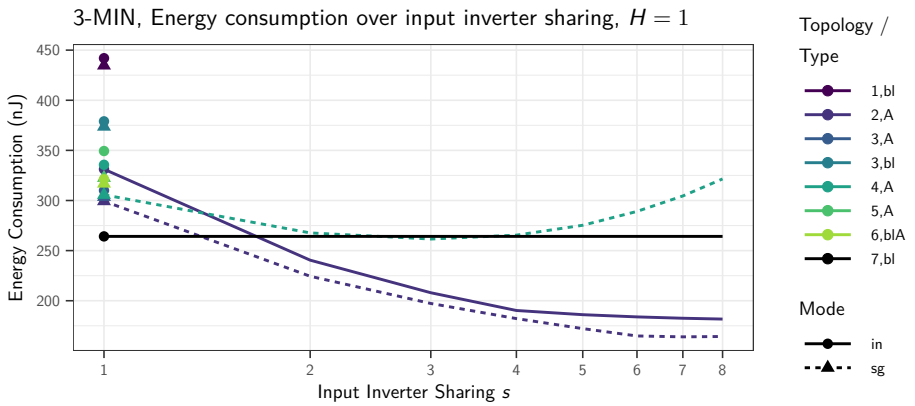


Figure 5.5: 3-MIN worst-case energy consumption. When input inverters can be shared with other circuits, i. e. their energy consumption distributes over more circuits, reconfigurable implementations become more energy-efficient than the baseline static variant.

with a 31 ps and 36 ps delay, respectively.

Input Inverter Sharing

As I wrote earlier, another benefit of the reconfigurable variants is that they expose the input inverters that they use to implement reconfiguration. These can be shared in a larger design, reducing the overall effort that is needed to produce this input signal as was shown for reconfigurable circuits in [42]. In VLSI designs it is a common occurrence that inverted signals are produced as part of the larger function. They are also produced as part of buffers that are needed to meet signal quality demands and slew rate constraints. So, whenever this happens, the reconfigurable implementations may provide an energy and power advantage over the static implementation.

The next experiment quantifies dynamic energy consumption of the reconfigurable variants when the input inverters are shared. Modelling and experiment parameters are kept the same as for the $(E_{\max}, t_{\max})_{\mathsf{A,B,C}}$ experiment and Figure 5.5 reproduces its results for an inverter sharing factor of 1.

The assumption is that sharing the inverter with another logic gate means that only 50 % of the energy consumption of the inverter is counted towards the 3-MIN circuit. Each step on the x-axis corresponds to the number of circuits that *each* input inverter is shared between. So, the minimum factor is 1, because each used inverter is at least connected to the circuit under test. The energy for each

circuit variant c and inverter sharing factor s is computed as follows:

$$E_{\max}(c,s) = \max\left(E^{\text{iso}}(c) + \sum_{i \in \text{Inv}_c} \frac{1}{s} \times E(i, s-1)\right)$$

where Inv_c is the set of input inverters used by circuit variant c, and $E(i, s-1)$ is the dynamic energy consumption of input inverter i that is connected to circuit c and $s-1$ additional inverter loads, i. e. shared between s circuits.

Obviously, each input inverter drives a higher load and gets slower the more it is shared between circuits. So, depending on how this inverter is used inside the 3-MIN variant, sharing may eventually increase dynamic energy consumption as a result of a very slow slew rate.

Figure 5.5 shows only those reconfigurable variants that fall below the value of $E_{\max}(7\text{,bl,in},\cdot) = 264\,\text{nJ}$ of the static implementation, which is represented by a black horizontal line. The static implementation does not employ input inverters and is, thus, unaffected by sharing. The points at the left show the values $E^{\text{pwr}}_{\max}$ from Figure 5.4 for each variant, repeating that without input sharing, no reconfigurable variant is more energy-efficient than the static implementation in the worst case.

The NAND/NOR topology of variants 2,A,* proves to be beneficial regarding inverter sharing, outperforming the static variant for all succinct cases. Variant 1,bl,sg (not shown) follows the same curve shifted up by almost 150 nJ. Looking back at Figure 5.1, we can see that it has almost the identical structure as 2,A,sg. Just the inputs are connected differently. This asymmetry is, what gives variant 2,A,sg the advantage. In this experiment, I enforce that all input inverters are shared between s circuits, much to the disadvantage of variant 1,bl,sg. It would perform much better in an asymmetric setting, in which only one of the input inverters is shared while the others can deliver their power exclusively to the circuit.

The only other variant breaking the static energy consumption is variant 4,A,sg. While this variant starts as low as variant 2,A, it cannot benefit as much from inverter sharing and is more susceptible to high input loads. All remaining implementations follow this curve, exceeding their initial energy consumption for very high inverter loads. For these implementations, the energy consumption of the input inverters start to dominate the product term in the energy equation.

It can be observed that the *single* mode proves to be the more energy-efficient implementation. Its exact effect depends on multiple variables like the actual transient in the worst-case switching event and the threshold voltage difference between Schottky barrier gates and inner gates. This explains why the two modes (dashed and solid curves) of the same variant are sometimes closer together or further apart and do not exactly follow the same trend.

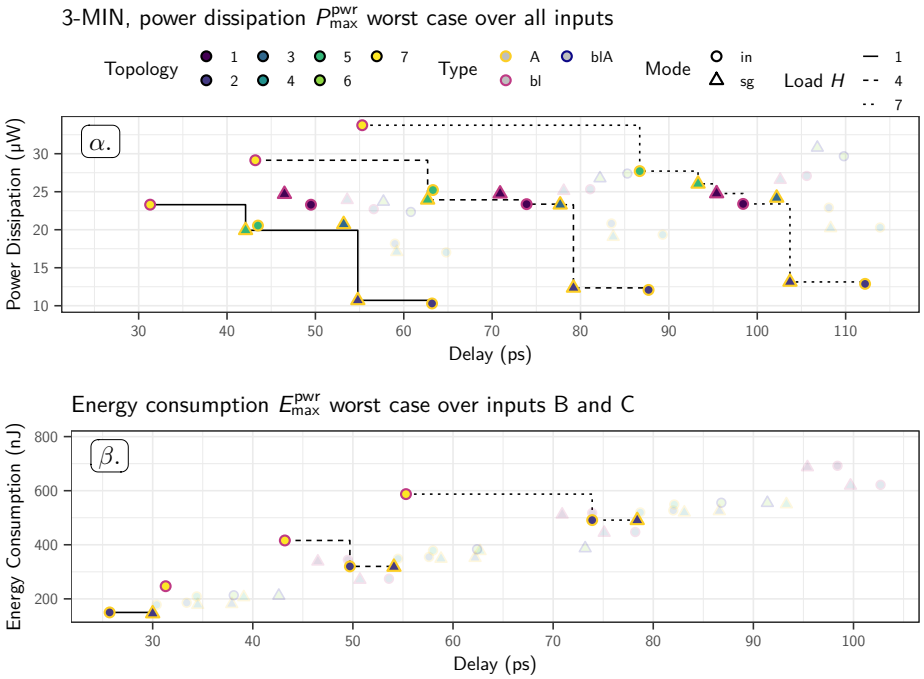


Figure 5.6: 3-MIN worst-case analysis for multiple output loads.

Output Load Sensitivity

The experiments so far analysed the 3-MIN function for an output load $H = 1$. This made sense, given that a high output load shadows the internal characteristics of a circuit that stem from its topology and input connection pattern. For a general judgement about the suitability of a particular implementation in a specific application, its load-dependent behaviour is an important information.

This experiment is conducted with the same modelling parameters as before. Input inverters are considered as internal parts of the circuit. Additionally, the circuits are quantified for multiple output loads $H = 1, 4, 7$. This experiment shows the development of the circuit performance across realistic output loads. For this experiment, I concentrate on those circuits that will end up on the Pareto front for at least one of the output load configurations. They are shown in the usual shapes and colours while the suboptimal implementations are shown greyed out for reference. As detailed in Figure 5.4, in the worst case over all inputs, the static variant 7,bl,in solely dominates the energy consumption performance. Thus, Figure 5.6 shows $(P_{\max}, t_{\max})_{\mathsf{A,B,C}}$ at the top, but it shows $(E_{\max}, t_{\max})_{\mathsf{B,C}}$ at the bottom.

Both graphs show three sets of results, one for each output load value, result-

ing in three distinct Pareto fronts. The first observation from the $P_{\max}$ graph (α) is, that with rising load more and more circuits become eligible implementation choices. This means that many of the reconfigurable circuits are usable in the right environment. The ever growing distance between the static implementation and the other circuits in the group shows that the small design of the reconfigurable circuits makes them very susceptible for output load changes. This is commonly known from other primitive circuit implementations and can be usually addressed with transistor scaling. There is also a growing distance between the static implementation 7,bl,in and the reconfigurable variants along the vertical axis, meaning that the static implementation is most susceptible for output load changes regarding power consumption. This is best seen in case of variants 1,bl,* (dark blue/purple). For $H = 1$, they are not Pareto-optimal. They are still highlighted, because both are Pareto-optimal for $H = 7$, becoming even more power efficient than variants 6,A,* (green/yellow).

The lower graph β, that displays $E_{\max}$ over $t_{\max}$ for inputs B and C, clearly shows that even when the only thing that is known is, that one input is used significantly less often than the other two (effectively resulting in a reconfiguration scenario), variants 2,A,* outperform the static implementation for small output loads and stay competitive across the range of output loads.

5.2 Analysis of 3-Input Exclusive OR Gate

Another interesting function is the 3-input exclusive OR function. It is known to be critical in arithmetic operations and cryptography because of its perfect symmetry. So, there is neither a dominant input variable nor, being a non-unate function, a dominant input value. Every single-input switch triggers an output change regardless of the current input combination. This property results in a large set of 19 principle variants and a total of 49 circuit implementations in:

8 topologies

6 reconfiguration types A, B, C, balanced with preference for A or C (bl/A, bl/C) and fully balanced (bl)

3 reconfiguration modes single, inner and transmission gate mode.

The 3-XOR function is not primitive, in the sense, that there exists no static complementary implementation of a single logic stage with three *independent* inputs. All implementations as CMOS circuits require two stages one way or another. They either implement the function from two cascaded circuits (e. g. two 2-XOR circuits) or resolve input dependencies by inverting input signals, which also sends each affected signal through two logic stages. This means, that there is no obvious benefit for the static implementation anymore, which, in case of the 3-MIN circuit, could avoid using input inverters at all. The structure of the Boolean function also enables the use of the third transistor reconfiguration mode, the *transmission*

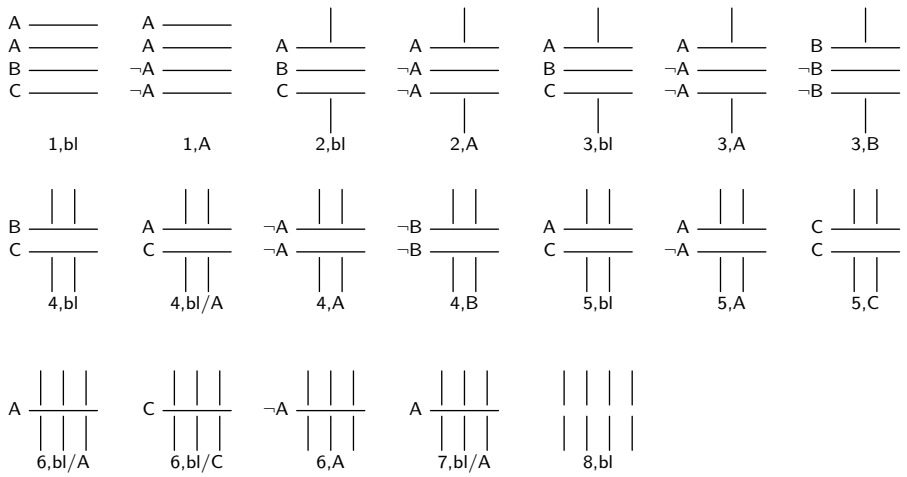


Figure 5.7: The 19 principle variants of 3-XOR as stick diagrams. They result in 49 implementations.

gate mode. Not all Boolean functions allow implementations in this mode. See Table 2.3 on page 33 for a recapitulation of the effects of each mode.

These conditions hold for all principle variants except the static implementation 8,bl,in. The 19 principle variants are shown as stick diagrams in Figure 5.7. It can be seen that they are generally similar in structure to the 3-MIN principle variants. Some topologies show a wider range in the number of reconfiguration types that they can implement. Topology 3, 4, 5 and 6 can favour different input signals without repeating themselves in the static part, creating implementations with slightly different characteristics. One thing they have in common is, that they need all three input signals direct and inverted, with few exceptions; variants 1,bl,tg, 5,A,tg and 5,C,tg, only need two input signals inverted. Due to this, reconfigurable circuits show real benefits over the static implementation, also in worst-case scenarios.

The two most interesting implementations are two transmission gate mode variants 1,A,tg and 1,bl,tg shown in Figure 5.8. They are special for various reasons. Variant 1,A,tg was first described in [65] and was found via hand-crafting. When redrawn as a complementary transistor circuit, with the two transistors sourced by A going on one side and the rest going on the opposite side, variant 1,A,tg is a natural extension of 3-MIN 1,A (the NAND/NOR variant). An additional limitation in [65] came from the fact that the two Schottky barrier gates were connected with each other and had to be used in unison. At the time, it was lucky circumstance that transistor devices that have to be used in an X,Y,X configuration are sufficient to implement 3-XOR without producing short-circuits. This variant is special, because it manages to strike a good balance between the loads

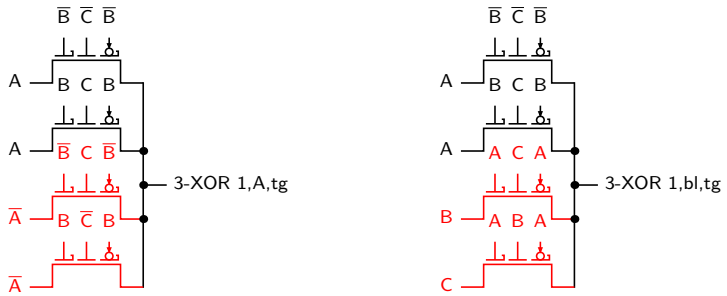


Figure 5.8: 3-XOR implementations that outperform the static implementation in the worst-case. Variant 1,A,tg was discovered by [65] while the other was found in this DSE.

of each input signal despite preferring the single input signal A to reconfigure:

$$H(\mathsf{A}) = H_{\text{out}} + 1 \qquad H(\mathsf{B}) = 3 \qquad H(\mathsf{C}) = 2$$
$$H(\overline{\mathsf{A}}) = H_{\text{out}} \qquad H(\overline{\mathsf{B}}) = 2 \qquad H(\overline{\mathsf{C}}) = 1$$

Since $H_{\text{out}} \geq 1$, all inputs are within 2 inverter equivalents of each other. (Hereinafter, I sometimes use H_{out} to highlight the attached output load when computing *some* load H in inverter equivalents for *some* point in the circuit. Where in most places, I refer to the output load just as H.) Additionally, because the inverted signals suffer a delay penalty anyhow, their lower loads relative to their direct counterparts contribute to an optimal delay performance of that variant for small output loads.

The second implementation that stands out, is variant 1,bl,tg. Figure 5.8 highlights that only two devices actually change, but these changes have significant consequences. By connecting signals B and C to the source contacts, this implementation can avoid using the inverter for signal $\overline{\mathsf{A}}$ altogether. The changed loads on the input signals are as follows:

$$H(\mathsf{A}) = H_{\text{out}} + 2 \qquad H(\mathsf{B}) = H_{\text{out}} + 2.5 \qquad H(\mathsf{C}) = H_{\text{out}} + 2$$
$$H(\overline{\mathsf{A}}) = 0 \qquad H(\overline{\mathsf{B}}) = 1 \qquad H(\overline{\mathsf{C}}) = 0.5$$

As can be seen, the load of all inputs depends on the output load now. The missing load from input inverter $\overline{\mathsf{A}}$ is compensated by using A as the reconfiguration input for both transistors, again leading to a balanced load distribution across the inputs. The unconditional output load dependence is slightly detrimental to its worst-case delay performance, but losing one inverter, which reduces the number of transistors by 20 %, has an enormous effect on power dissipation and energy consumption.

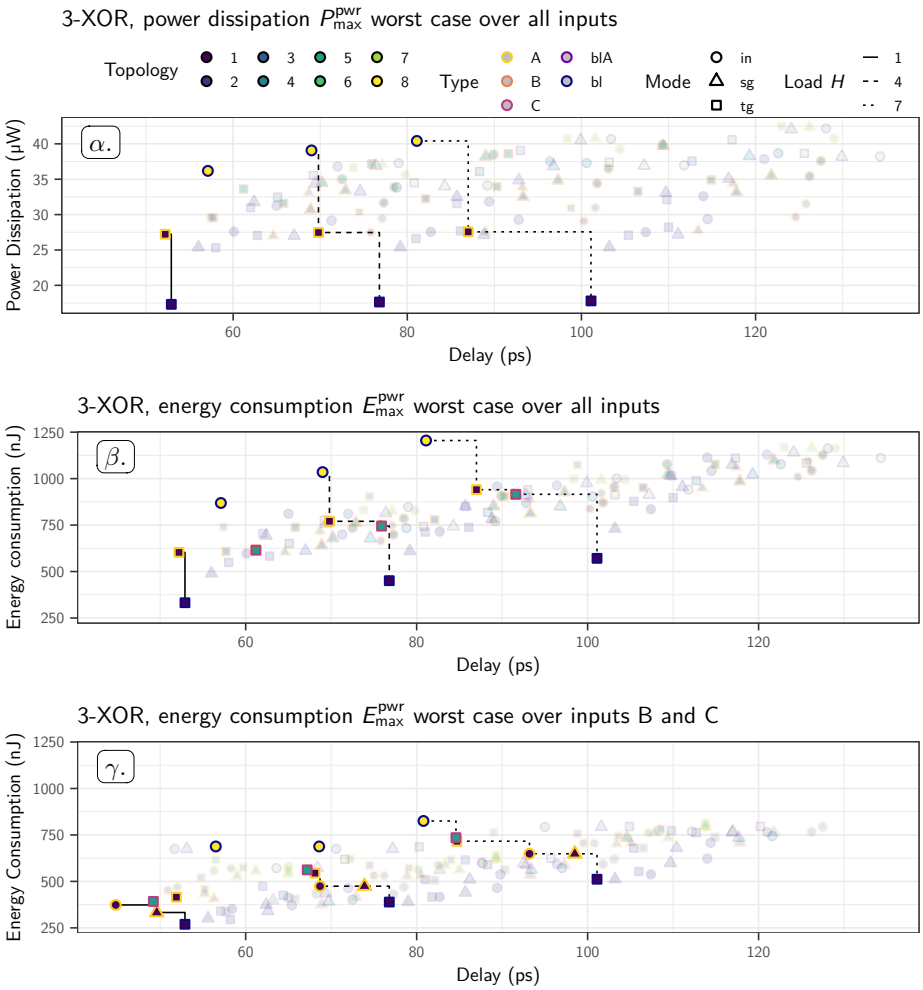


Figure 5.9: 3-XOR worst-case analysis for multiple output loads.

5.2.1 Worst-Case Analysis

The worst-case analysis of the 3-XOR circuit variants is performed as described in Section 5.1.2. The performance metrics for the circuits in isolation, $E_{\max}^{\text{iso}}$ and $P_{\max}^{\text{iso}}$, are not shown or further considered in this thesis.

The results for varying output loads $H = 1, 4, 7$ can be seen in Figure 5.9 on page 111.

For small output loads $H < 4$, the static variant 8,bl,in (yellow/blue circle) is no longer Pareto-optimal. Variant 1,A,tg (dark blue/yellow square) and 1,bl,tg outperform the static implementation in all three measures $t_{\max}$, $P_{\max}$, $E_{\max}$. For $H = 1$, delay is reduced from 57 ps to 52 ps, a 9 % improvement, while energy consumption per operation can be reduced down to 38 %.

While the reconfigurable variants are still more susceptible to output load increase, which is a result of their small size with merely one or two active transistors in any switching event, variant 1,A,tg can hold up for good, making it a competitive option in conditions with a reduced output load.

In case of measuring $(E_{\max}, t_{\max})_{\mathsf{B,C}}$ (γ), the results become even more pronounced. Many more variants, some of which are only shown shaded, because they are not Pareto-optimal, can outperform the static implementation for small loads, and the static implementation only becomes competitive for output loads greater than $H = 4$. As before with the 3-MIN function, topology 1 dominates the field, with variant 1,A,in being the fastest variant with 45 ps and with variant 1,A,sg being slightly slower but more energy efficient than the former, beaten only by variant 1,bl,tg. The transmission gate mode variants are usually the least power efficient implementations of their respective principle variant. Due to the setup of their input connections, they are somewhere in between the inner mode that also uses the reconfiguration signal at both Schottky barrier gates and the single mode; because they can reduce the number of gates by connecting one of the variables of the min-term that implements a particular transistor solely to the source contact and to none of the gates.

Independence of worst-case states The worst-case energy consumption experiment $(E_{\max}, t_{\max})_{\mathsf{A,B,C}}$ (β) shows one additional circuit on the Pareto front, variant 5,C,tg (green/yellow square). This is interesting, because it sits in an unlikely sweet spot, that makes it a viable candidate under energy consumption requirements, but which is not also Pareto-optimal under power dissipation, although both measures are related by the circuit delay. The reason for this peculiar phenomenon is that all the three worst-case quantities result from pairwise different switching events. Model checking not only returns the results directly but also provides witness states that are causal to the obtained numbers. In this case, the witness states are those states in the program graph of the experiment model and query, at which the circuit is still in equilibrium but the input automaton has already decided on the next input combination. By virtue of constraining the input automaton to only generate new input transitions when the circuit is

in equilibrium, combined with the queries considering only those input switches that also effect an output transition, these witness states are exactly the set of interesting states. So, the witness states contain all necessary information to directly infer the switching event in terms of logic inputs that switch from one value to another, although the model itself mostly speaks about voltages and currents. For this circuit, the model checker returned the following state transitions:

$$St\left(t_{\max}^{5,\mathrm{C},\mathrm{tg}}\right) = \{(0^*, 0^*, 0^*)\} \qquad \begin{aligned} St\left(P_{\max}^{5,\mathrm{C},\mathrm{tg}}\right) &= \{(1^*, 1^*, 1)\} \\ St\left(E_{\max}^{5,\mathrm{C},\mathrm{tg}}\right) &= \{(0^*, 1^*, 0^*)\} \end{aligned}$$

St is the function that returns the set of logic state transitions that caused the quantified result. The notation (a^*, b^*, c^*) represents the ordered vector of circuit inputs A, B and C, where an asterisk marks an input that is about to flip. Thus, $(0^*, 0^*, 0^*)$ describes the state transition $(0, 0, 0) \rightarrow (1, 1, 1)$. This means that, because maximum delay neither occurs during maximum power dissipation nor during maximum energy consumption, it must be better for the latter two switching events. For $E_{\max}$, it is good enough to reduce energy consumption down to a Pareto-optimal level.

This shows, that it is not good enough to start from a delay-driven simulation to find the worst-case delay $t_{\max}$ and assume that $P_{\max}$ and $E_{\max}$ result from the same switching event. Not only are the states before the event pairwise different, but so are the states after the switching event. Hence, the events are not even adjacent to each other in any testing scenario.

Understanding the $E_{\max}$ Outlier

The results of experiment $(E_{\max}, t_{\max})_{\mathrm{B,C}}$ (Fig. 5.9 γ) show another noteworthy outcome. For an output load $H \in \{1, 4\}$, the static variant 8,bl,in (yellow/blue circle) consumes exactly the same amount of energy per operation in the worst-case; $E_{\max}^{H=1} = 687.88\,\mathrm{nJ}$ and $E_{\max}^{H=4} = 687.99\,\mathrm{nJ}$. This means, that either dynamic energy consumption is independent of output load, or the causal states are so unusual that energy consumption inside the circuit dominates the overall energy budget in the switching event. Again, the model checker returns witness states that cause the observed behaviour:

$$\begin{aligned} St\left(E_{\max,H=1}^{8,\mathrm{bl},\mathrm{in}}\right) &= \{(1, 0^*, 0^*)\} \\ St\left(E_{\max,H=4}^{8,\mathrm{bl},\mathrm{in}}\right) &= \{(1, 0^*, 0^*)\} \end{aligned} \qquad St\left(E_{\max,H=7}^{8,\mathrm{bl},\mathrm{in}}\right) = \left\{ \begin{array}{l} (1, 0^*, 0\ ), \\ (1, 0\ , 0^*), \\ (0, 1\ , 0^*), \\ (0, 0^*, 1\ ) \end{array} \right\}$$

From the topology of the circuit (cf. Fig. 5.7), we can see that high short-circuit currents are more likely when multiple input signals switch simultaneously, because the circuit features many parallel branches in the pull-up and pull-down networks. This is the case for $H = 1$ and $H = 4$, where inputs B and C switch from $0 \rightarrow 1$. What is interesting about these state transitions, is, that they

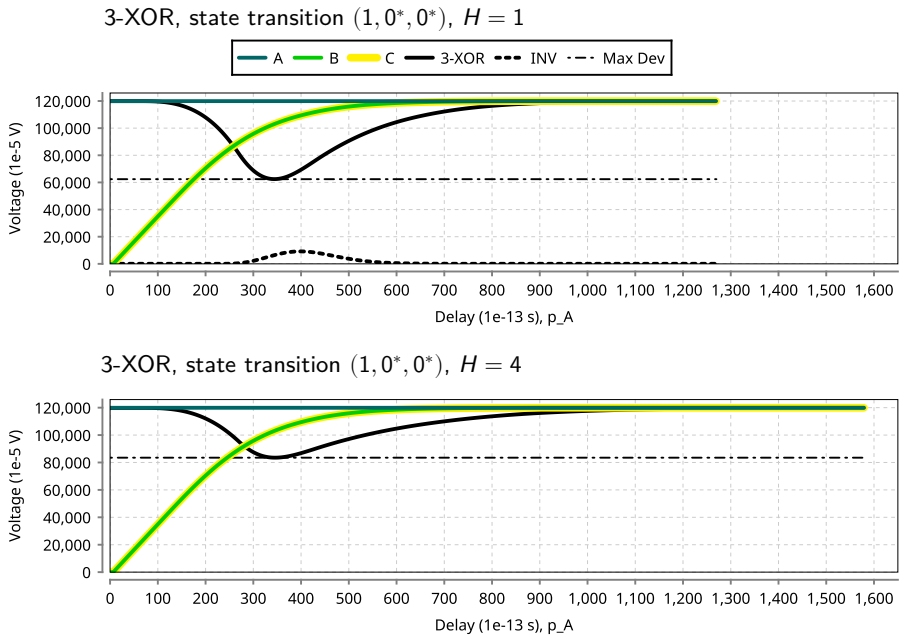


Figure 5.10: Simulation of switching events that cause $E_{\max}^{8,\mathrm{bl,in}}$ for $H \in \{1, 4\}$. Results are in model space resolution.

should not change the output at all. Since, the 3-XOR implements the odd parity function, i. e. the function that determines whether the number of ones in the input are odd, switching a pair of inputs must result in the same output value as before, because switching a pair of inputs can never affect the parity of the input set (hence the name of the function). For $H = 7$, these are no longer the causal states, but switching either signal B or C from $0 \to 1$ causes the highest dynamic energy consumption. Remember that signal A cannot switch in this experiment, but also see that it is part of the set of causal states in both polarities. This hints at the output load being dominant for larger loads regardless of what happens in the other two cases.

But what happens in the cases for $H = 1$ and $H = 4$? Starting from the causal model states, I simulated the state transitions. As the causal states are the states right after the input automaton has selected a new target input state, there is no other non-deterministic choice involved. The simulation follows a single path along the program graph of the experiment until the circuit is in equilibrium again. Thus, the resulting input and output traces can be directly extracted from the model checker and are shown in Figure 5.10. It shows that input A remains constant 1 and that the signals B and C switch from $0 \to 1$ as described by the witness states. In both cases, the output (black) deviates significantly in time

and magnitude before settling back at 1.2 V. Both sets of traces end when the circuit is back in equilibrium, which, for $H = 1$, is after 1269 steps (≈ 127 ps) and for $H = 2$, is after 1578 steps or ≈ 158 ps.

Looking back at how this circuit is modelled as a charge transport network, each transistor in circuit 8,bl,in directly connects one of the power supply lines with the output. No reconfiguration takes place, nor are inputs fed directly to the output via pass gates. This means, that there are no chains of charge storage nodes, but the network consists (apart from the power supplies) of a single charge storage node at the output of the circuit. Because the capacitance attributed to the node is constant, the voltage changes represented by the black curves reflect the equivalent of all charges that contribute to the energy consumption of the circuit. So, the area under both curves must be identical (because the consumed energy is also the same), and the difference in magnitude and extent comes from the difference of capacitance attributed to the charge storage node. The capacitances compute to the following inverter equivalents:

$$H(x) = x + p \times H^{\mathrm{p}}_{\mathrm{chan}} + n \times H^{\mathrm{n}}_{\mathrm{chan}} \quad \text{with } H^{\mathrm{p}}_{\mathrm{chan}} = H^{\mathrm{n}}_{\mathrm{chan}} = \frac{1}{2}$$

$$H(1) = 1 + \frac{8}{2} = 5 \qquad H(4) = 4 + \frac{8}{2} = 8$$

where 8 comes from the eight transistor drain contacts being connected to either 1 or 4 inverter equivalents at the output. The proportion of $^{8}/_{5}$ of the output load corresponds exactly to the proportion of the magnitudes of the deviations of the output from its normal value (see Max Dev in Figure 5.10) with $^{576\,\mathrm{mV}}/_{365\,\mathrm{mV}}$. This gives further confidence in the precision of the model and its quantitative results.

These deviations can be clearly problematic, because the transistor device model defines the threshold voltages of the Schottky barrier gates to be $V_{\mathrm{th}} = 400$ mV and for the inner gates to be even lower with $V_{\mathrm{th}} = 200$ mV. Depending on the transmitted amount of energy, both output hazards may be sufficiently large to trigger a spurious switching event in the next logic stage. To check this, I slightly altered the experiment shown in top of Figure 5.10 for $H = 1$ with respect to the one below. Instead of attaching a capacitance equivalent to one inverter, I attached an actual standard-sized inverter of the same technology to the output of the 3-XOR circuit. This inverter is itself loaded with $H = 1$. The dashed black line shows that the inverter reacts to the output hazard but is not triggered to switch over.

Output hazards Using another model checking query, I can enumerate those state transitions that do not ultimately trigger a circuit output change but which lead to output hazards that exceed a threshold of 400 mV, corresponding to V_{th} at the Schottky barrier gate. This query is built similarly to the ones used to determine $t_{\max}$, $P_{\max}$ and $E_{\max}$:

Hazard$_{34}$ For all paths on which the inputs are not *stable* (i. e. in transition), and which eventually encounter an output transition that is either 34 % above the minimum output voltage of 0 V or 34 % below the maximum output voltage of 1.2 V, do the following: Enumerate the causal states that are obtained by a *switch* but that do not *effect*$_o$ for all outputs o.

Of course, the query is independent of the particular circuit and works the same for any number of inputs and outputs. For multiple outputs, the query enumerates all state transitions affecting any output, but adapting the query to filter transitions for a particular output can be done by fixing *effect*$_o$ to a particular output o.

The 3-XOR has 8×8 possible state transitions of which exactly half actually change the output. As output hazards can only occur during state transitions that should *not* affect the output, the number of possible state transitions in which output hazards can occur is 32.

Table 5.1 shows all circuits and their state transitions that cause output hazards larger than 400 mV, queried with *Hazard*$_{34}$. The shown results range from one to six affected state transitions across 24 affected variants, which result in the hazard ratio of hazardous transitions over possible hazardous state transitions. Hence, the hazard ratio can change in steps of 3.125 % for the 3-XOR circuits. It reaches its maximum with 18.75 % in the static variant 8,bl,in, but the median of circuits is only affected in 6.25 % of the transitions. One other detail in this table is, that all circuits that produce output hazards, do so by switching two inputs in the same direction, with one exception. Variant 2,A,in produces the hazard when two inputs switch in opposite directions.

Only 24 out of 49 total variants produce large hazards. Closer inspection of Table 5.1 reveals that four principle variants are not affected at all: 1,bl, 2,bl, 4,bl/A and 5,bl. This means, the reasonably fast and exceptionally power-efficient variant 1,bl,tg does not produce serious output hazards.

5.2.2 Functional Verification

This section is a small deviation, because it concerns itself with a 2-XOR circuit instead of 3-XOR. It is done, to show an interesting case for the use of quantitative analysis for functional verification, and it aligns well with the surrounding sections, because two 2-XOR circuits in series are often used to implement the 3-XOR function using standard CMOS devices. Apart from the strong quantification possibilities, model checking and formal methods in general are predestined to be used for functional verification. This topic has not been touched so far, because the way the DSE algorithm constructs the circuits, guarantees that they are *functionally correct*. In this thesis, a circuit works *functionally correct* if and only if for every *switch* that *effect*$_o$, has an effect on output o, the output transitions to the correct output value that is specified by its equivalent Boolean function eventually. The limit at which the Boolean output value is considered to be reached is defined to be ± 200 mV from the ideal voltages V_{SS} / V_{DD}, which represent the

Table 5.1: 3-xor variants and state transitions that exhibit output hazards that are greater than 0.4 V. Enumerated with $Hazard_{34}$.

Variant v	Affected transitions $St(v)$	Hazard ratio
1,A,sg	$\{ (1^*,1,1^*), (0^*,0,0^*) \}$	6.25 %
1,A,tg	$\{ (1^*,1,1^*), (0^*,0,0^*) \}$	6.25 %
2,A,in	$\{ (0^*,1^*,0), (0^*,0,1^*) \}$	6.25 %
3,A,sg	$\{ (1^*,1,1^*), (1^*,1^*,0), (1^*,0,1^*), (0^*,1,0^*), (0^*,0^*,1) \}$	15.625 %
3,A,tg	$\{ (1^*,1,1^*), (0^*,0,0^*) \}$	6.25 %
3,bl,tg	$\{ (0^*,0^*,1) \}$	3.125 %
3,B,in	$\{ (1^*,0,1^*), (0^*,1,0^*) \}$	6.25 %
3,B,sg	$\{ (1^*,0,1^*), (0^*,1,0^*) \}$	6.25 %
3,B,tg	$\{ (1^*,1^*,0), (1^*,0,1^*), (0^*,0^*,1) \}$	9.375 %
4,A,in	$\{ (1^*,1^*,0), (1^*,0,1^*), (0^*,1,0^*), (0^*,0^*,1) \}$	12.5 %
4,B,in	$\{ (1^*,0,1^*), (0^*,1,0^*) \}$	6.25 %
4,B,sg	$\{ (1^*,0,1^*), (0^*,1,0^*) \}$	6.25 %
4,B,tg	$\{ (1^*,0,1^*), (0^*,1,0^*), (0^*,0^*,1) \}$	9.375 %
5,A,in	$\{ (1^*,0,1^*), (1,0^*,0^*), (0,1^*,1^*), (0^*,1,0^*) \}$	12.5 %
5,A,tg	$\{ (1,0^*,0^*), (0,1^*,1^*) \}$	6.25 %
5,C,in	$\{ (1^*,1^*,0) \}$	3.125 %
6,A,in	$\{ (1,0^*,0^*), (0,1^*,1^*) \}$	6.25 %
6,A,tg	$\{ (1,0^*,0^*), (0,1^*,1^*) \}$	6.25 %
6,bl/A,in	$\{ (0,1^*,1^*) \}$	3.125 %
6,bl/A,sg	$\{ (0,1^*,1^*) \}$	3.125 %
7,bl/A,in	$\{ (1,0^*,0^*), (0,1^*,1^*) \}$	6.25 %
7,bl/A,sg	$\{ (1^*,1^*,0), (1^*,0,1^*), (1,0^*,0^*), (0,1^*,1^*) \}$	12.5 %
7,bl/A,tg	$\{ (1^*,1^*,0), (1^*,0,1^*), (0,1^*,1^*), (0^*,1,0^*) \}$	12.5 %
8,bl,in	$\{ (1^*,1^*,0), (1^*,0,1^*), (1,0^*,0^*), (0,1^*,1^*), (0^*,1,0^*), (0^*,0^*,1) \}$	18.75 %

logic values 0/1. The output may assume arbitrary values arbitrarily often before finally coming to rest in the correct voltage range. The query that verifies functional correctness must obviously depend on the circuit under test, specifically on the Boolean function it implements.

For this experiment, I selected a complementary CMOS implementation of the 2-XOR function that was found via an evolutionary algorithm and demonstrated in [63]. Its circuit diagram is shown in Figure 5.11. Two aspects make it interesting to investigate. First, it was generated under circumstances similar to this thesis. In both cases, a DSE method was employed to generate new circuit implementations. While the DSE algorithm in this thesis is known to produce correct results, it is also clear that it has serious shortcomings. Its implementation method is severely self-limited to implementing actual min-terms in each transistor, neglecting transistor sharing completely. Additionally, its underlying method is known to have exponential complexity, making it known to fail quickly for a growing number of inputs. The evolutionary algorithm used in [63], promises better scaling behaviour at the expense of losing correctness. Second, the circuit implementation resulting from the evolutionary construction does not work in all cases, at least not with the device I have used, although it is a production device.

With model checking this is directly verifiable and the offending state transitions are enumerable. The query that verifies the functional correctness of a 2-XOR circuit is defined as follows:

func On all paths, the following proposition must hold for all states which satisfy $stable_I$:

$$\left(V_\mathsf{A} < \frac{V_\mathrm{DD}}{2} \wedge V_\mathsf{B} < \frac{V_\mathrm{DD}}{2}\right) \vee \left(V_\mathsf{A} > \frac{V_\mathrm{DD}}{2} \wedge V_\mathsf{B} > \frac{V_\mathrm{DD}}{2}\right) = V_\mathrm{out} < 0.2\,\mathrm{V} \bigwedge$$
$$\left(V_\mathsf{A} < \frac{V_\mathrm{DD}}{2} \wedge V_\mathsf{B} > \frac{V_\mathrm{DD}}{2}\right) \vee \left(V_\mathsf{A} > \frac{V_\mathrm{DD}}{2} \wedge V_\mathsf{B} < \frac{V_\mathrm{DD}}{2}\right) = V_\mathrm{out} > 0.8\,\mathrm{V}$$

It is obvious that this proposition can be generated automatically from the Boolean function and from knowledge about the experimental setup. Once the circuit has reached equilibrium on its current input stimulus, the experiment setup ensures that the input voltages are either charged to V_DD or V_SS. Due to how the input stimuli are generated from inverters, these voltages are not reached with numerical precision. So instead of testing for matching voltages, it suffices to check whether the input voltages are in the upper or lower half of the domain. As stated earlier, the output is expected to deviate less than 200 mV from its intended voltage for the circuit to be considered functionally correct.

For the circuit shown in Figure 5.11, the query *func* determined that it is not correctly operating in all state transitions. Figure 5.12 shows a trace for the following seven state transitions:

$$(0^*, 0)^\dagger \to (1^*, 0) \to (0, 0^*)^\dagger \to (0^*, 1)^\dagger \to (1, 1^*)^\dagger \to (1^*, 0) \to (0, 0^*)^\dagger$$

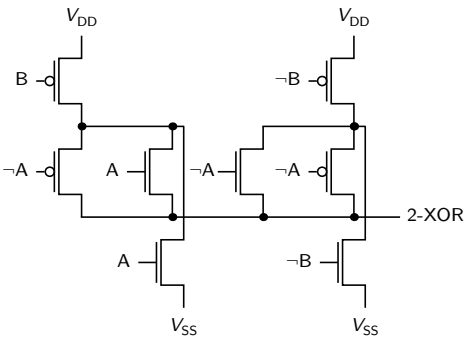


Figure 5.11: 2-XOR design resulting from an evolutionary search shown in [63].

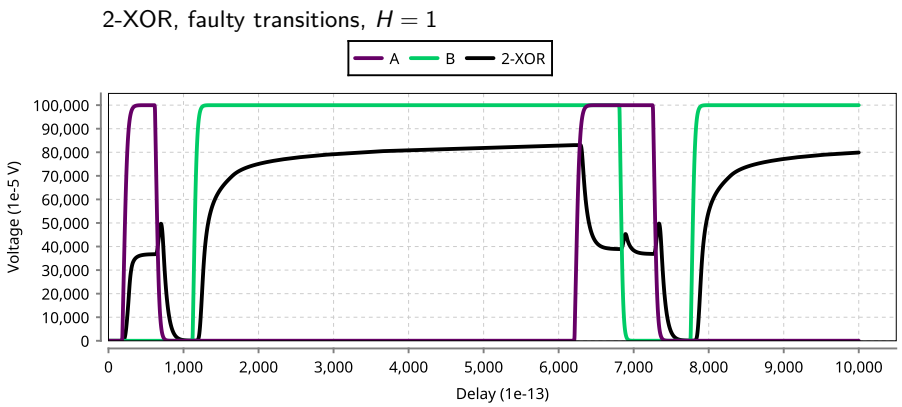


Figure 5.12: 2-XOR circuit from [63]. Implemented with the 32 nm CMOS transistor `prism-gen` model from the device shown in [41]. The device operates at a nominal supply voltage of 1 V. Results are in model space resolution.

The trace shows, that the circuit only works correctly for state $(0, 0)$ which also resolves the stable states with a partially charged output. Transitions to non-functional states are marked with a dagger (†). As explained in the previous chapter, the input automaton does not end the charging of the output prematurely. The circuit actually is in equilibrium with the partially charged output seen in the trace, before the input automaton selects the next transition non-deterministically.

For completeness sake, the experiment was set up with the same modelling parameters as earlier experiments in this chapter, only the supply voltage and the transistor device were changed and the input automaton was simplified to switch only one input at a time. The device used to implement this circuit is the model I used for studying the charge transport network modelling precision in

the previous chapter. This device is a model of a 32 nm CMOS production device shown in [41]. Its performance and characteristics can be trusted the most of the `prism-gen` models, because it has been shown to work the same compared to its commercial SPICE models. As is often seen in CMOS devices, they do not feature equal PMOS and NMOS performance. So, the shown circuit needs considerable transistor scaling not only to optimise its performance but to make it work at all.

The quantitative formal analysis not only adds a verification of the circuit functionality on the electrical level, but it also incorporates the specifics of the used transistor devices. So, even while the circuit worked for the device that was used by its original authors, it is worthwhile to have a method to verify its functional correctness on new devices, as well.

5.2.3 Probabilistic Analysis

Long-Run Average Quantification

Despite the usual metrics to characterise circuit performance being focused on the worst-case, the circuit, obviously, spends most of its time in its average cases. For asynchronous logic, the average case is the defining metric (apart from necessary consideration of catastrophic worst-case behaviour) for circuit performance. On the one side asynchronous logic is not well regarded in the eye of the wider scientific public, because it is expensive to build. It needs more transistors than synchronous logic, is provably unable to achieve the same peak performance (both because handshake signals contain actual information while clock signals do not) and is hard to synthesise from abstract circuit descriptions (a bad synthesis results in performance penalties, not misbehaviour). On the other side, on average, it is much more power efficient and has a better delay performance, because it does not have to scale its clock frequency back to meet the worst case at all times. While delay performance is immediately perceived from surrounding actors, energy efficiency is only perceived in large quantities that are averages across a large set of operations, which is true also for synchronous logic. This is a domain where probabilistic model checking has a lot to offer for circuit characterisation.

In this experiment, which was first shown by myself and Steffen Märcker in [47] for the 3-MIN function and in its current form in [48], I investigate the average circuit performance of the 3-XOR function for two different sets of input switching probabilities. The modelling parameters and the experiment setup are the same as in the previous experiments with the following differences:

- Each circuit is loaded with $H = 1, 4, 7$ inverter equivalents.
- The quantified delay t_{avg} is the long-run average delay. It describes the average delay that the circuit would exhibit, when all switching events would be tried ad infinitum according to a fixed probability distribution that determines the likelihood that any circuit inputs switch or not.

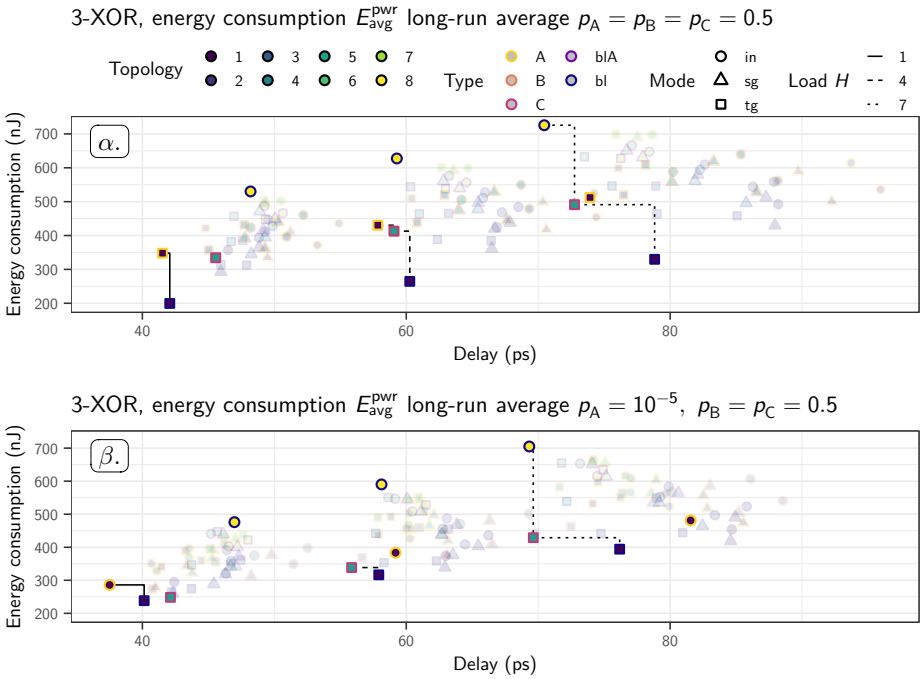


Figure 5.13: 3-XOR long-run average analysis for multiple output loads.

- The quantified dynamic energy consumption E_{avg} describes the long-run average energy consumption according to the same method as t_{avg}. E_{avg} over t_{avg} is quantified for two sub-cases:
  - Long-run average for an input probability distribution of $p_A = p_B = p_C = 1/2$, called $(E_{\text{avg}}, t_{\text{avg}})_{0.5}$
  - Long-run average for an input probability distribution of $p_A = 10^{-5}$, $p_B = p_C = 1/2$, called $(E_{\text{avg}}, t_{\text{avg}})_{10^{-5}}$

Please note that the input switching probabilities are independent of each other and, thus, do not add up. Their probability merely describes how likely it is that a particular input switches in the next switching event. Thus, for all switching events to be equally likely to occur in the long-run, each input must switch with probability $1/2$ and not $1/3$ as one might be tempted to think.

The results of this experiment are shown in Figure 5.13. As before, they highlight the Pareto-optimal variants grouped by output load. The lumps of circuits are closer together now, as their average performance is naturally closer together than their performance at the extremes. Nevertheless, the general trend across the implementations remains clearly visible, the reconfigurable circuits are

more susceptible to increasing output load, weakening their delay performance considerably. The static variant cannot gain from small a output load, but is least affected by its increase.

Notably, many reconfigurable variants outperform the static implementation 8,bl,in for $H = 1$. For the reconfigurable scenario, in which signal A is switched considerably less often, it is the overwhelming majority. Additionally, contrary to the worst case, all reconfigurable variants are always more energy-efficient than the static variant, regardless of the input switching probability or output load.

The two best-performing reconfigurable variants 1,A,tg and 1,bl,tg are also Pareto-optimal in many cases. While variant 1,bl,tg, dominates the energy consumption performance due to its saving of a third input inverter, variant 1,A,tg shows a mixed performance. For the average experiment $(E_{\text{avg}}, t_{\text{avg}})_{0.5}$, this variant does perform very well at first, but is superseded by variant 5,C,tg for high loads. In the reconfiguration scenario $(E_{\text{avg}}, t_{\text{avg}})_{10^{-5}}$, this variant also performs exceptionally well on average, proving to be very stable and almost able to keep up with the static variant up to $H = 7$. It, too, saves one inverter and strikes a seemingly good balance between reconfiguration and a powerful drive that sustain high output loads. Specifically, it avoids inverting signal C, giving it a performance advantage in the reconfiguration scenario, where, on average, half of the switches are performed with the fast signal C. On top of that, variant 5,C,tg does not suffer from large output hazards. This seems to be generally true for the transmission gate mode variants, as can be seen for $H = 7$. All variants that are close behind the Pareto-front are transmission gate variants that can make use of this reconfiguration mode, which does not force the source signal to be also used as a reconfiguration signal on the same transistor, reducing the load of that input.

Parametric Delay

So far, the figures showed that the delay of the reconfigurable variants varied with the change of output load and with the activity of signal A. The model checker is able to quantify the average delay depending on the switching probability of signal A. This is not achieved by sampling, but the model checker computes the average delay as a function with probability p_{A} as its parameter. How this is done, is detailed in the work of Steffen Märcker [34]. Though, one important precondition is that the probability distribution to enter a certain Boolean state must be the same as the probability distribution to leave it to the following state. The circuit models that are discussed in this chapter fulfil this precondition. With this parametric delay function determined for a particular circuit, circuit designers are now able to directly compute the expected average delay. Without any application knowledge, they would just use $p_{\mathsf{A}} = 0.5$ and would arrive at the same values that were shown in the upper graph of Figure 5.13. It is now clear, that the lower graph in that figure, displays another point on the curve for $p_{\mathsf{A}} = 10^{-5}$. Using the function, any point can be computed without the need to run the whole model checker on a specific set of parameters again.

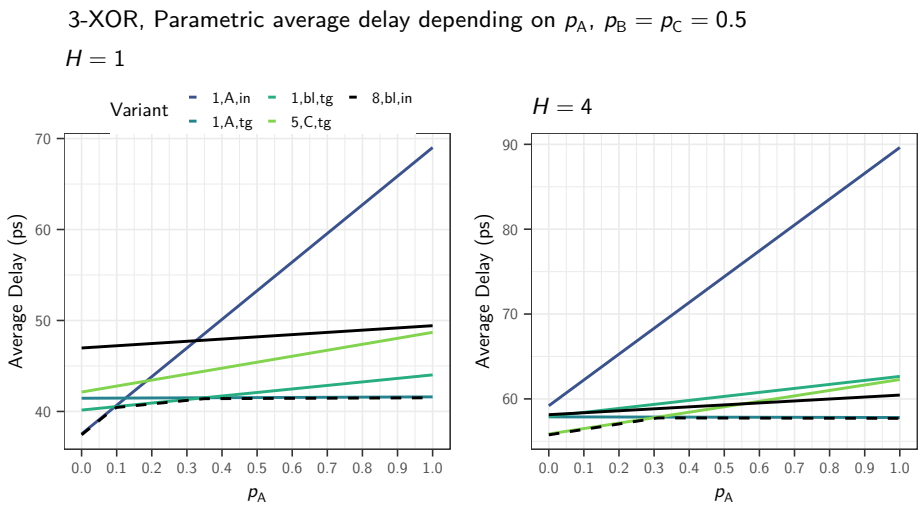


Figure 5.14: 3-XOR parametric average delay analysis for multiple output loads.

For the 3-XOR function, I have computed the parametric delays for a range of circuit implementations. The results are shown in Figure 5.14 for all variants that are at least once Pareto-optimal in Figure 5.13. The graphs show the resulting rational functions, not sampled data points, for switching probabilities $p_B = p_C = 0.5$. While the left graph shows the functions for each circuit for output load $H = 1$, the right graph shows the set of functions for output load $H = 4$. For small output loads, variant 1,A,in can be the fastest variant in a known reconfiguration scenario, in which signal A is used for reconfiguring the circuit. Just below a 10 % probability of switching A variant 1,bl,tg becomes better on average. The optimum curve is highlighted with a black dashed line and follows the best-performing variant. It switches over a second time to variant 1,A,tg at around 35 % probability. The graph also shows, that variant 1,A,tg is largely unaffected by the switching probability of signal A and that multiple candidates exist that are better than the static implementation 8,bl,in shown in black.

On the right side for $H = 4$, the picture gets different. Multiple variants, 1,A,tg, 1,bl,tg and 8,bl,in, have almost the same average delay when signal A does not switch. Again, variant 1,A,tg turns out to perform unaffected by the switching probability of A, still outperforming the static implementation. Yet, below 30 % switching probability, variant 5,C,tg performs best, outperforming the static implementation at around 55 % probability. The high susceptibility to output load shows for variant 1,A,in, which is unable to compete against any other variant for $H = 4$.

For asynchronous designs, parametric average delay performance would be a valuable addition to the synthesis library. In complex designs, input switching probabilities can be very well computed, because the individual signals are by

no means independent. Data is usually transmitted in words and their encoding allows for simple approximations upfront. Also, the results show, that the approximations can be rough to allow for an improved circuit selection.

In this chapter, we have seen the automatic design space exploration and quantitative analysis at work for two highly-relevant logic gates. It turns out, that the analysis needs to look at various metrics under multiple experiment conditions to convey the complex performance results (good and bad) that come with the various reconfigurable circuit implementations. One important outcome was that performance results for different metrics may come from different circuit input transitions in different circuit states. Thus, the exhaustive analysis proved fruitful in that it cannot miss the correct result or mislead the experimenter in the analysis. The results of new analyses like parametric average delay and output hazards would be useful for down-the-road use in electronic design automation (EDA) processes, because they convey functional aspects of standard-cell behaviour beyond delay and power dissipation that could lead to overall better circuit designs. Also, the effects of hazards on the following logic stage was investigated. Lastly, the nature of formal methods also enabled us to map an uncommon 2-XOR circuit implementation to a new device and verify (or, here, disprove) its functionality.

Chapter 6

Conclusion and Future Work

In this thesis, I establish the notion of transistor reconfiguration that needs to be separated from the common understanding of reconfiguration as used for field-programmable logic circuits. Chapter 2 motivates my research with hand-crafted logic gates and the establishment of the fundamental effects of the polarity-controllable transistor that enable reconfiguration and their interplay inside particular circuits. It further introduces the design space that the technology provides with its reconfiguration modes and its balanced and biased reconfiguration types.

Chapter 3 compares the reconfigurable standard cells both on a structural level and in terms of absolute delays. By comparing circuit implementations across two transistor device technologies, one polarity-controllable lab technology and one established CMOS production device, it serves to show the complexity in evaluating the viability of reconfigurable circuit designs. Although reconfigurable implementations are structurally beneficial, they do not necessarily show indisputable performance improvements, warranting closer and broader investigation. In this chapter, I show the abilities of reconfigurable standard cells to serve in *explicit* reconfiguration and *implicit* reconfiguration. I develop and improve a small ALU and optimise a well-known conditional sum adder design by replacing sub-structures with reconfigurable standard cells without altering the circuit's function. The improved arithmetic circuits benefit from the compact standard cell designs and especially excel in low-power scenarios while at the same time, their functional enhancement shows potential to increase timing performance by reducing logic stages.

As a result from these insights, I develop a transistor technology evaluation methodology in Chapter 4 that specialises in early technology evaluation. Its focus is to research digital circuit designs using reconfigurable transistors that are at a very early design stage. The methodology allows easy access to device parameters and allows the creation of new transistor devices with little effort and only few input data points. Additionally, circuit and experiment description are carried out in the same input language, which allows device and/or circuit parameters to

change as part of an automatic experiment series. This allows a researcher to home in on wanted device properties that need the development focus. Although using the same input language, device description, circuit design and experiment setup are separated use cases that allow the description of devices and circuits independent from each other or a particular experiment. Nevertheless, they also allow the re-use of experiment setups for multiple circuits. The whole methodology is facilitated by modelling the electrical transistor network as a charge transport network that is computable by the probabilistic model checker PRISM. It incorporates the transistors and capacitive nodes, which represent voltages and cause currents, as well as input conversion from logical inputs to voltages. The chapter also shows the use of generalised model checking queries that enable the analysis of particular measures of interest, like delay performance or energy consumption, that are independent of the circuit under test. A design space exploration algorithm that generates a set of circuit variants from a Boolean input function and its implementation completes the methodology and allows an automated analysis of a range of circuit variants over a multiple quantities and experiment variants. The transistor device models are compared against TCAD data and the resulting logic gates are compared with simulations made with commercially available SPICE models. They demonstrate the accuracy of the device and charge transport network models.

In Chapter 5, I use this methodology and the tooling that comes with it to conduct an extensive performance analysis of two reconfigurable circuits, the 3-input minority logic gate and the 3-input exclusive OR gate. The Chapter presents the complexity of judging the performance of reconfigurable circuits, as it depends highly on the application scenario. It shows that reconfigurable circuit variants can outperform static implementations especially in situations of strongly biased inputs. They are representative for user-level reconfiguration scenarios, in which a particular function is selected over long periods of time. Circuit reconfiguration is shown to provide benefits by allowing input inverters to be shared amongst standard cells, further reducing complexity and necessary energy budget. The methodology proves to be precise in identifying the causing circuit states and transitions that decide on a circuit's performance. It shows that especially worst-case results of different measures may be caused by different state transitions of the same circuit. Some worst-case results are even caused by hazardous transitions that might have gone unnoticed in simulations, because they arguably do not belong to the usual input testing set. Using the formal analysis approach, the hazards can be automatically analysed traced back to the causing states after the fact, explaining the underlying circuit behaviour. Additionally, circuits are analysed regarding their general tendency to exhibit hazards, which is important for certain asynchronous logic schemes like NULL convention logic, because it may affect the correctness of its operation. The ability to verify the correctness of a circuit is used to investigate a 3-XOR design, proposed by another group and found via an evolutionary search, with an available state-of-the-art CMOS device. It is found to be very susceptible to transistor device performance and does not

work with unscaled transistor devices. My research concludes with a probabilistic and parametric analysis of reconfigurable 3-XOR circuit variants and it shows that the performance gains that can be achieved with reconfigurable circuits on average are much higher than the worst-case results suggest. This is especially interesting for energy consumption and power dissipation, because these measures are only relevant on larger scales where all parts do never exhibit their worst-case behaviour at the same time. Additionally, short but large deviations from the average quantity can be tolerated without immediately sacrificing functional correctness. Thus, a worst-case quantity that almost never happens might not bear any significance to the application scenario.

6.1 Future Work

The proposed modelling and quantitative analysis framework is focused on early device characterisation and its design space exploration targets reconfigurable circuit variants created from polarity-controllable transistors. In this regard, the modelling and analysis is completely device-independent and new devices like carbon nanotube or 2-D material devices like carbon nanoribbon FETs, single-layer MoS_2 transistors [16, 44] or tunnelling FETs [6, 25, 26] can be added. They would be modelled as closed form functions that reproduce the voltage transfer characteristics and that depend solely on contact voltages and momentary currents as described in Chapter 4.

For practical reasons, this framework targets the PRISM model checker as its computational basis, for it is the only tool capable of computing the parametric long-run average measures and the only one providing the witness states of its quantitative results. Another valuable compute platform would be the STORM model checker, as it has a modern code base supporting multiple compute cores, though it is at least lacking the aforementioned properties at the time of this writing.

Further automated processing of the design space exploration results is obvious. They can be ranked and categorised and serve as input data to improved process design kits that are based on the measured transistor technology. This would make the new-found standard cells immediately useful for experimentation in practical VLSI circuit applications. The intended process design kit's characterisation conditions have to be replicated yielding the required worst-case delay results and output load-dependent slow-down factors used by current EDA libraries. As these conditions are independent of the specific logic gate, circuit characterisation and conversion can be fully automated as well.

Beyond this immediate application, parametric averages of the measured quantities could be a valuable addition to the metrics that go into the selection process used by EDA design tools. The resulting rational functions could be used in enhanced PDKs that can guide the standard cell selection depending on known input switching probability distributions. This could lead to a substantial improvement

in average performance, without unduly sacrificing worst-case performance. It would especially accommodate asynchronous circuit design automation which is much more dependent on average performance numbers.

Lastly, this framework could serve as an early feedback to semiconductor research. By performing an early analysis on a default set of standard cells, researchers gain quick insight into the absolute and relative performance of the lab device compared to established or other researched technologies. The support for multiple device technologies within the same experiment setup enables researchers to investigate the compatibility of new devices with established transistors. This would be beneficial when the new device is targeted as a back-end technology to support a faster device technology used for high-power, high-performance applications. The freedom to describe new input drivers and modify transistor devices also supports the research of transistors that are used to act as sensors (light, chemicals). Additionally, the charge transport network model naturally replicates the analogue transient behaviour of the modelled devices and, thus, can be used to produce early insights into the function of the researched technology. This also extends to memory technology devices like memristors or ferroelectric devices that exhibit stateful behaviour. The network model naturally supports feedback loops and stateful elements.

One conceivable extension to the approach would be to construct well-defined abstractions that capture the quantitative and, as much as necessary, the qualitative results of the detailed analyses of individual standard cells. These abstractions could be composed to quantify larger circuits or standard cells in more complex experiment setups saving on the model state space while retaining the quality of the experiment results. Another use case of this extension would be to use probabilistic model checking to determine realistic application-dependent input probabilities for standard cells of interest. They can, in turn, be used to compute realistic long-run average performance data to further improve combinational circuits by placing the right reconfigurable standard cell implementation at the right locations in the circuit. Integrating all this data into EDA tools and PDKs would make these insights usable to the enhanced tool flows that also optimise average performance.

Appendix A

Notational conventions

This document uses the following notational conventions to describe syntactic and semantic elements of the `prism-gen` domain-specific language.

In general, source code is displayed in `typewriter` font both in separate listings and within paragraphs. `prism-gen` is a language with strong relations to CommonLisp and uses its typical list notation and formatting style. This means that the closing parenthesis are written after the end of the last element even in lists that span multiple lines and not on a line of their own (as is common in curly-brace languages like C). In multi-line lists, lines starting from the second line are indented as follows:

- by two characters if the first line contains only a single word,
- to the beginning of the second list element or
- to a later element if that maintains the inner semantic of the list.

```
(constant
  T_SCALE
  1e12)
(input-node drv_A
            :slope 5)
(cap-net-node output :stable T
                     :nodes (drv_A))
```

Comments are introduced by one or more semicolons and they are typeset in *italic font*. Additionally, the phrase "Example:", on a line of its own, is used to separate the definition of a language construct from an example of its use.

This document uses the Backus Naur Form (BNF) to describe the grammar of language constructs. To maintain the visual resemblence of the final language, grammar descriptions use the previously explained list style and augment them with a change of font or visual elements to convey the necessary information.

`name` Denotes a terminal symbol or a piece of code that is written as-is. These are keywords of the language or literal values.

`name` Denotes a non-terminal symbol, which can be the name of a parameter or value or another BNF rule that is explained elsewhere.

Modified BNF Syntax

The main extension to regular BNF syntax is the following:

`⟦`*`X`*`⟧` An expression of this form describes that a list *`X`* is to be spliced into the surrounding list and that its elements can appear in any order.

`X` must be of the following form:

`X1` `|` `...` `|` *`Xn`* Each element *`Xi`* can have the form *`Y`*, *`Y`*`*` or `{`*`Y`*`}`^{1}.

The expression `⟦`*`X`*`⟧`, thus, means that a list of the form

`(`*`Xi1`* `...` *`Xin`*`)` $1 \leq n$ is spliced into the surrounding expression, such that if $j \neq k$ and $1 \leq j, k \leq n$, then either *`Xij`* $\neq$ `X`*`ik`* or *`Xij`* $=$ *`Xik`* $=$ *`Xv`*, where for some v, $1 \leq v \leq k$, *`Xv`* has the form *`Xv`*`*`. This means an element is only allowed to appear more than once if it has the form *`Xv`*`*`. Additionally, an element *`Xij`* must appear if it has the form `{`*`Xv`*`}`^{1}.

`⟦`*`X`*`⟧+` This notation adds the additional restriction that at least one of the elements in *`X`* must be used.

For example, the expression

```
(x ⟦A | B* | {C}¹⟧ y)
```

means that at most one `A` may appear in the resulting list, any number of `B`'s and exactly one `C`. The following are valid applications of this BNF rule:

```
(x C y)
(x A C y)
(x C A y)
(x B C y)
(x C B y)
(x A B C y)
(x B A C y)
(x B C A y)
(x C B A y)
(x C A B y)
(x A B B C y)
```

Appendix B

`prism-gen` Programming Interfaces

Charge Storage Nodes

A charge storage node *node* provides the following programming interface:

$$
\begin{aligned}
V_{node} &\rightarrow \texttt{V\_}\textit{node} \\
C_{node} &\rightarrow \texttt{C\_}\textit{node}\ \text{(capacitive nodes only)} \\
D_{node} &\rightarrow \begin{cases} \textit{node}\texttt{\_i} & \text{(voltage sources)} \\ \textit{node}\texttt{\_chrg} & \text{(dirac and integrator nodes)} \end{cases} \\
\text{Bool}(\textit{node}) &\rightarrow \textit{node}\texttt{\_d} \\
\Delta D_{node} &\rightarrow \text{(represented via update value } D'_{node}\text{)} \\
D_{node} &\rightarrow \textit{node}\texttt{\_next}\ \text{(new value after state update)} \\
V_{node} &\rightarrow \texttt{V\_}\textit{node}\texttt{\_next}\ \text{(dto.)} \\
\textit{stable}_{node} &\rightarrow \texttt{stable\_}\textit{node}
\end{aligned}
$$

Charge Transport Nodes

There is no common interface to the internal of charge transport nodes. Parametrisation is usually done on a device level where the device in question is a single object that understands the documented set of parameters. Nevertheless, each device provides a set of connection points that are inherent to the context of the device. The `rfet-node` transistor has a different set of connectors, named after the gates and channel contacts, than a simple resistor. So, their interfaces can be

looked up in the corresponding sections in Chapter 4 or the program documentation that comes with `prism-gen`.

Bibliography

[1] Qurat Ain and Osman Hasan. "Formal Timing Analysis of Digital Circuits". In: vol. 1008. Communications in Computer and Information Science. Springer, Jan. 2019, pp. 84–100.

[2] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. "Biconditional BDD: A novel canonical BDD for logic synthesis targeting XOR-rich circuits". In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, DATE 2013*. IEEE, Mar. 2013, pp. 1014–1017.

[3] Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. "Approximate Symbolic Model Checking of Continuous-Time Markov Chains". In: *Proceedings of the 10th International Conference on Concurrency Theory, CONCUR '99*. Ed. by Jos C. M. Baeten and Sjouke Mauw. Vol. 1664. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999, pp. 146–161.

[4] Jan C. Bioch and Toshihide Ibaraki. "Decompositions of Positive Self-Dual Boolean Functions". In: *Discrete Mathematics* 140.1–3 (1995), pp. 23–46.

[5] John M. Birkner and Hua-Thye Chua. *Programmable array logic circuit*. US Patent 4,124,899. Nov. 1978.

[6] Stefan Blawid, Denise L. M. de Andrade, Sven Mothes, and Martin Claus. "Performance Projections for a Reconfigurable Tunnel NanoFET". In: *IEEE Journal of the Electron Devices Society* 5.6 (Nov. 2017), pp. 473–479.

[7] Dominique Borrione, Philippe Georgelin, and Vanderlei Moraes Rodrigues. "Symbolic simulation and verification of VHDL with ACL2". In: *System-on-Chip Methodologies & Design Languages*. Ed. by Peter J. Ashenden, Jean P. Mermet, and Ralf Seepold. Springer, 2001, pp. 59–69.

[8] William S. Carter, Khue Duong, Ross H. Freeman, Hung-Cheng Hsieh, Jason Y. Ja, John E. Mahoney, Luan T. Ngo, and Shelly L. Sze. "A User Programmable Reconfigurable Logic Array". In: *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*. CICC '86. May 1986, pp. 233–235.

[9] Kuo-Hsing Cheng and Shun-Wen Cheng. "Improved 32-bit Conditional Sum Adder for Low-Power High-Speed Applications". In: *Information Science and Engineering, Journal of* (2006), pp. 975–989.

[10] Edmund M. Clarke and E. Allen Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In: *Proceedings of the Workshop on Logics of Programs*. Ed. by Dexter Kozen. Vol. 131. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1981, pp. 52–71.

[11] Michele De Marchi, Davide Sacchetto, Stefano Frache, Jian Zhang, Pierre-Emmanuel Gaillardon, Yusuf Leblebici, and Giovanni De Micheli. "Polarity control in double-gate, gate-all-around vertically stacked silicon nanowire FETs". In: *2012 International Electron Devices Meeting*. IEEE. 2012, pp. 8–4.

[12] William C. Elmore. "The transient response of damped linear networks with particular regard to wideband amplifiers". In: *Journal of applied physics* 19.1 (1948), pp. 55–63.

[13] Karl M. Fant and Scott A. Brandt. "NULL Convention Logic^{TM}: A Complete And Consistent Logic For Asynchronous Digital Circuit Synthesis". In: *Proceedings of the International Conference on Application Specific Systems, Architectures and Processors, ASAP '96*. IEEE, 1996, pp. 261–273.

[14] Peter Feldmann and Roland W. Freund. "Efficient linear circuit analysis by Padé approximation via the Lanczos process". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14.4 (1995), pp. 639–649.

[15] Gianluca Fiori and Giuseppe Iannaccone. "Simulation of Graphene Nanoribbon Field-Effect Transistors". In: *IEEE Electron Device Letters* 28.8 (Sept. 2007), pp. 760–762.

[16] Gianluca Fiori and Giuseppe Iannaccone. "Simulation of Graphene Nanoribbon Field-Effect Transistors". In: *IEEE Electron Device Letters* 28.8 (Aug. 2007), pp. 760–762.

[17] Limor Fix. "Fifteen Years of Formal Property Verification in Intel". In: ed. by Orna Grumberg and Helmut Veith. Berlin, Heidelberg: Springer, 2008, pp. 139–144.

[18] Vojtěch Forejt, Marta Z. Kwiatkowska, and David Parker. "Pareto Curves for Probabilistic Model Checking". In: *Proceedings of the 10th International Symposium on Automated Technology for Verification and Analysis, ATVA 2012*. Ed. by Supratik Chakraborty and Madhavan Mukund. Berlin, Heidelberg: Springer, 2012, pp. 317–332.

[19] Philippe Georgelin, Pierre Ostier, and Dominique Borrione. “A framework for VHDL combining theorem proving and symbolic simulation”. In: *Proceedings of the 3rd International Workshop on the ACL2 Theorem Prover and its Applications, ACL2 2002*. Grenoble, France, Apr. 2002, pp. 1–15.

[20] Gary D. Hachtel and Fabio Somenzi. *Logic synthesis and verification algorithms*. Springer, 2006.

[21] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. “Probabilistic Reachability for Parametric Markov Models”. In: *Proceedings of the 16th International SPIN Workshop on Model Checking Software, SPIN 2009*. Ed. by Corina S. Pasareanu. Vol. 5578. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 88–106.

[22] Robert F. Hartmann, Sau-Ching Wong, Yiu-Fai Chan, and Jung-Hsing Ou. *Programmable logic array device using EPROM technology*. US Patent 4,617,479. Oct. 1986.

[23] André Heinzig. “Entwicklung und Herstellung rekonfigurierbarer Nanodraht-Transistoren und Schaltungen”. PhD thesis. Technische Universität Dresden, Germany, 2015.

[24] Khaza Hoque, Otmane Ait Mohamed, Yvon Savaria, and Claude Thibeault. “Probabilistic model checking based DAL analysis to optimize a combined TMR-blind-scrubbing mitigation technique for FPGA-based aerospace applications”. In: *Proceedings of the 12th ACM/IEEE International Conference on Formal Methods and Models for Co-Design, MEMOCODE 2014*. IEEE, Oct. 2014, pp. 175–184.

[25] Adrian M. Ionescu and Heike Riel. “Tunnel field-effect transistors as energy-efficient electronic switches”. In: *Nature* 479 (Nov. 2011), pp. 329–337.

[26] Jae Won Jeong, Young-Eun Choi, Woo-Seok Kim, Jee-Ho Park, Sunmean Kim, Sunhae Shin, Kyuho Lee, Jiwon Chang, Seong-Jin Kim, and Kyung Kim. “Tunnelling-based ternary metal-oxide-semiconductor technology”. In: *Nature Electronics* 2 (July 2019), pp. 307–312.

[27] T. I. Kirkpatrick and N. R. Clark. “Pert as an Aid to Logic Design”. In: *IBM Journal of Research and Development* 10.2 (Mar. 1966), pp. 135–141.

[28] Tillmann Krauss, Frank Wessely, and Udo Schwalke. “Electrically reconfigurable dual metal-gate planar field-effect transistor for dopant-free CMOS”. In: *2016 13th International Multi-Conference on Systems, Signals & Devices (SSD)*. IEEE. 2016, pp. 681–686.

[29] Thomas Kropf, ed. *Formal Hardware Verification - Methods and Systems in Comparison*. Vol. 1287. Lecture Notes in Computer Science. Springer, 1997.

[30] Jayanand Asok Kumar, Lingyi Liu, and Shobha Vasudevan. "Scaling probabilistic timing verification of hardware using abstractions in design source code". In: *Proceedings of 2011 Formal Methods in Computer-Aided Design, FMCAD 2011*. IEEE, Oct. 2011, pp. 196–205.

[31] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. "Parametric probabilistic transition systems for system design and analysis". In: *Formal Aspects of Computing* 19.1 (2007), pp. 93–109.

[32] Mats G. Larson and Fredrik Bengzon. *The Finite Element Method: Theory, Implementation, and Applications*. Vol. 10. Texts in Computational Science and Engineering. Springer, 2013.

[33] Gaotian Lu, Yang Wei, Xuanzhang Li, Guangqi Zhang, Guang Wang, Liang Liang, Qunqing Li, Shoushan Fan, and Yuegang Zhang. "Reconfigurable Tunneling Transistors Heterostructured by an Individual Carbon Nanotube and MoS2". In: *Nano Letters* 21.16 (2021), pp. 6843–6850.

[34] Steffen Märcker. "Model Checking Techniques for Design and Analysis of Future Hardware and Software Systems". PhD thesis. Dresden: Technische Universität Dresden, Mar. 2021.

[35] Edward J. McCluskey. "Minimization of Boolean functions". In: *The Bell System Technical Journal* 35.6 (Nov. 1956), pp. 1417–1444.

[36] Carver Mead and Lynn Conway. *Introduction to VLSI systems*. Addison-Wesley, 1980.

[37] Rebeca Moura, Niels Tiencken, Sven Mothes, Martin Claus, and Stefan Blawid. "Reconfigurable NanoFETs: Performance Projections for Multiple-Top-Gate Architectures". In: *IEEE Transactions on Nanotechnology* 17.3 (May 2018), pp. 467–474.

[38] Laurence W. Nagel and Donald Oscar Pederson. *SPICE (Simulation Program with Integrated Circuit Emphasis)*. Tech. rep. EECS Department, University of California, Berkeley, Apr. 1973.

[39] Gethin Norman, David Parker, Marta Z. Kwiatkowska, and Sandeep K. Shukla. "Evaluating the Reliability of NAND Multiplexing with PRISM". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.10 (Oct. 2005), pp. 1629–1637.

[40] A. Odabasioglu, M. Celik, and L.T. Pileggi. "PRIMA: passive reduced-order interconnect macromodeling algorithm". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17.8 (1998), pp. 645–654.

[41] Paul Packan et al. "High performance 32nm logic technology featuring 2nd generation high-k + metal gate transistors". In: *2009 IEEE Int. Electron Devices Meeting, IEDM '09*. Dec. 2009, pp. 1–4.

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

[43] Lawrence T. Pillage and Ronald A. Rohrer. "Asymptotic waveform evaluation for timing analysis". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 9.4 (1990), pp. 352–366.

[44] Branimir Radisavljevic, Aleksandra Radenovic, Jacopo Brivio, Valentina Giacometti, and Andras Kis. "Single-layer MoS_2 transistors". In: *Nature Nanotechnology* 6.3 (2011), pp. 147–150.

[45] Shubham Rai, Jens Trommer, Michael Raitza, Thomas Mikolajick, Walter M. Weber, and Akash Kumar. "Designing efficient circuits based on runtime-reconfigurable field-effect transistors". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.3 (2018), pp. 560–572.

[46] Michael Raitza, Akash Kumar, Marcus Völp, Dennis Walter, Jens Trommer, Thomas Mikolajick, and Walter M. Weber. "Exploiting Transistor-Level Reconfiguration to Optimize Combinational Circuits". In: *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2017.

[47] Michael Raitza, Steffen Maercker, Jens Trommer, André Heinzig, Sascha Klüppelholz, Christel Baier, and Akash Kumar. "Quantitative Characterization of Reconfigurable Transistor Logic Gates". In: *IEEE Access* 8 (2020), pp. 112598–112614.

[48] Michael Raitza, Steffen Märcker, Shubham Rai, and Akash Kumar. "Exploring Standard-Cell Designs for Reconfigurable Nanotechnologies: A Formal Approach". In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, DATE 2022*. IEEE. 2022, pp. 1–6.

[49] MarkWide Research. *MWR Global Complex Programmable Logic Device (CPLD) Marked Report 2021*. Link to report. 2021.

[50] Maximilian Reuter, Johannes Pfau, Tillmann A Krauss, Jürgen Becker, and Klaus Hofmann. "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 (2020), pp. 114–125.

[51] Maik Simon, B. Liang, D. Fischer, M. Knaut, A. Tahn, T. Mikolajick, and W. M. Weber. "Top-Down Fabricated Reconfigurable FET With Two Symmetric and High-Current On-States". In: *IEEE Electron Device Letters* 41.7 (2020), pp. 1110–1113.

[52] Gilbert Strang and Georg J. Fix. *An analysis of the finite element method*. Wellesley-Cambridge Press, 1973.

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

[54] Simon M. Sze. *Physics of semiconductor devices*. New York, NY: John Wiley & Sons, 1981.

[55] Technical Committee ISO/TC 20/SC 14. *ISO 16290:2013 Space systems – Definition of the Technology Readiness Levels (TRLs) and Their Criteria of Assessment*. `https://www.iso.org/standard/56064.html`. Nov. 2013.

[56] Jens Trommer. "Towards Reconfigurable Electronics by Functionality-Enhanced Circuits and Germanium Nanowire Devices". PhD thesis. Technische Universität Dresden, Germany, 2017.

[57] Jens Trommer, André Heinzig, Tim Baldauf, Thomas Mikolajick, Walter M. Weber, Michael Raitza, and Markus Völp. "Reconfigurable nanowire transistors with multiple independent gates for efficient and programmable combinational circuits". In: *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2016, pp. 169–174.

[58] Jens Trommer, André Heinzig, Uwe Mühle, Markus Löffler, Annett Winzer, Paul M. Jordan, Jürgen Beister, Tim Baldauf, Marion Geidel, Barbara Adolphi, Ehrenfried Zschech, Thomas Mikolajick, and Walter M. Weber. "Enabling Energy Efficiency and Polarity Control in Germanium Nanowire Transistors by Individually Gated Nanojunctions". In: *ACS Nano* 11.2 (2017), pp. 1704–1711.

[59] Vaibhav Vaidya, Jungbae Kim, Joshua N. Haddock, Bernard Kippelen, and Denise Wilson. "SPICE optimization of organic FET models using charge transport elements". In: *IEEE Transactions on Electron Devices* 56.1 (2009), pp. 38–42.

[60] C. H. Van Berkel, Mark B. Josephs, and Steven. M. Nowick. "Applications of asynchronous circuits". In: *Proceedings of the IEEE* 87.2 (Feb. 1999), pp. 223–233.

[61] Preeti Wadhwani and Shubhangi Yadav. *Field Programmable Gate Array (FPGA) Market Size, Competitive Market Share & Forecast 2021–2027*. Link to report. 2021.

[62] Walter M. Weber, Lutz Geelhaar, Andrew P. Graham, Eugen Unger, Georg Duesberg, Maik Liebau, Werner Pamler, Caroline Chèze, Henning Riechert, Paolo Lugli, and Franz Kreupl. "Silicon-Nanowire Transistors with Intruded Nickel-Silicide Contacts". In: *Nano Letters* 6.12 (2006). PMID: 17163684, pp. 2660–2666.

[63] Luděk Žaloudek and Lukáš Sekanina. "Transistor-Level Evolution of Digital Circuits Using a Special Circuit Simulator". In: *Evolvable Systems: From Biology to Hardware*. Ed. by Gregory S. Hornby, Lukáš Sekanina, and Pauline C. Haddow. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 320–331.

[64] Jian Zhang, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. "Dual-threshold-voltage configurable circuits with three-independent-gate silicon nanowire FETs". In: *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2013, pp. 2111–2114.

[65] Jian Zhang, Xifan Tang, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. "Configurable circuits featuring dual-threshold-voltage design with three-independent-gate silicon nanowire FETs". In: *Circuits and Systems I: Regular Papers, IEEE Transactions on* 61.10 (2014), pp. 2851–2861.

[66] Shuguang Zhao and Licheng Jiao. "Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm". In: *Genetic Programming and Evolvable Machines* 7.3 (2006), 195–210.

Terms & Abbreviations

(N)AND

(N)OR

3-MIN 3-input minority

3-XOR 3-input exclusive OR

FinFET

NoC network on chip

ALU arithmetic logical unit

AND

ASIC application-specific integrated circuit

BNF Backus Naur Form

CCA conditional carry adder

CG control gate

CGRA coarse-grain reconfigurable architecture

CLB configurable logic block

CMOS complementary metal-oxide semiconductor

CPLD complex programmable logic device

CPU central processing unit

CSV comma-separated values

CTL computation tree logic

CTL*

DFE dynamic function exchange

DNF

DPR dynamic partial reconfiguration

DSE design space exploration

DSL domain-specific language

DTMC discrete-time Markov chain

EDA electronic design automation

EPFL École polytechnique fédérale de Lausanne

FDSOI fully-depleted silicon on insulator

FEM finite element method

FET field-effect transistor

FPGA field-programmable logic array

GENW germanium nanowire

INV

INVZ

IP core intellectual property core

LTL linear time logic

LUT lookup table

MAJ

MDP Markov decision process

MIGFET multiple independent gate field-effect transistor

MIN

MOS metal-oxide semiconductor

MOSFET metal-oxide semiconductor field-effect transistor

MUX multiplexer

NAND

NFET n-channel field-effect transistor

NMOS

NMUX inverting multiplexer

NOR

OR

PAL programmable array logic

PCG polarity control gate

PCTL*

PDK process design kit

PFET p-channel field-effect transistor

PLA programmable logic array

PLD programmable logic device

PMC probabilistic model checking

PMOS

PRISM PRISM Model checker

PROM

RFET reconfigurable field-effect transistor

RTL register transfer level .

SB Schottky barrier

SINW silicon nanowire

SPICE simulation program with integrated circuit emphasis

SRAM static random access memory

TCAD technology computer-aided design

TIGFET three independent gate field-effect transistor

TRL technology readiness level

VHDL very large-scale integrated circuit hardware description language

VLSI very large-scale integrated

X(N)OR

XNOR equivalence

XOR antivalence