# ABSTRACT


Title of Dissertation: POLYMORPHIC CIRCUITS:
THE IDENTIFICATION OF POSSIBLE SOURCES
AND APPLICATIONS

Timothy Dunlap
Doctor of Philosophy, 2024

Dissertation Directed by: Professor Gang Qu
Department of Electrical and Computer Engineering

Polymorphic gates are gates whose function depends on some external or environmental conditions. While there has been research into both the creation and applications of polymorphic gates, much remains unknown. This dissertation, motivated by the recent security applications of polymorphic gates, seeks a systematic approach to generating polymorphic gates.Its contributions include a polymorphic interoperability framework, the first study on the source of polymorphism, time-based polymorphic gates, and polymorphism in the sub-threshold design.

Polymorphic circuits are commonly created with evolutionary algorithms [3]. Because the evolutionary algorithm operates in ways that are not always obvious, precise mechanisms of polymorphism are not immediately clear in the resulting gates and has not been reported before. This dissertation, for the first time, identifies multiple structures that impact the polymorphic nature of the gates, which sheds light on how to create polymorphic gates. This discovery is based on a categorization methodology that evaluates the quality of polymorphic gates and finds the robust

ones for further investigation of polymorphism. By combining the discovered structures with the evolutionary algorithm, high quality polymorphic gates can be generated faster as demonstrated in the subthreshold design domain.

Time-based polymorphism was discovered during the time analysis of evolved polymorphic circuits while searching for the sources of polymorphism. This occurs when the function of the circuit depends on the sample rate of the circuit and is based on some input combinations not quickly reaching the output they move towards. Therefore, when the circuit is running at different clock frequency, it may exhibit different functionality. This is time-based polymorphism.

As one application of polymorphic gates, this dissertation presents a framework that can enhance the fault coverage of any fault testing method by utilizing polymorphic gates. The proposed framework starts with any traditional fault testing approach, and when it becomes less effective in covering uncovered faults, it employs a gate replacement strategy to selectively replace certain standard logic gates by polymorphic gates of specific polymorphism. This concept is demonstrated in the dissertation with examples of a D flip-flop and the ISCAS85 C17 benchmark. This work has high practical value in subthreshold design where circuit manufacture defects increase significantly.

In summary, this dissertation presents multiple contributions to the study of polymorphic circuits. It discovers multiple sources of polymorphism and how the results of an evolutionary algorithm can be filtered into higher quality solutions. It also examines time-based polymorphism as a new form of polymorphism with security applications. Finally, an enhancement to stuck-at fault testing using polymorphic gates is presented. This allows for easier testing of corner-cases that are hard to detect using traditional methodologies and holds promise for improving the reliability of testing, particularly in the subthreshold domain.

# POLYMORPHIC CIRCUITS: THE IDENTIFICATION OF POSSIBLE SOURCES AND APPLICATIONS

by

Timothy Dunlap

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:
Professor Gang Qu, Chair/Advisor
Professor Alan L. Sussman, Dean's Representitive
Assistant Professor Cunxi Yu
Associate Professor Manoj Franklin
Associate Professor Timothy Horiuchi

# Dedication

To my family and friends. You are my strength, and you mean the world to me.

# Acknowledgments

I would like to thank everyone who has made this dissertation possible and who has supported me through graduate school. It has been a long journey, and I truly could not have done it without every one of you.

I am deeply indebted to Professor Gang Qu for his invaluable guidance and unwavering support throughout my doctoral journey, and it was quite the journey. His profound knowledge and insightful feedback were instrumental in shaping my research and refining my academic voice. I am particularly grateful for his patience and encouragement as I faced both the difficulties of part-time graduate school and the COVID pandemic and during the other challenging moments of this endeavor. Professor Gang Qu has been a true mentor and source of inspiration, and I am honored to have had his guidance on this path.

To Bill Johnson, and the Laboratory of Physical Sciences as a whole, I offer my thanks for bringing this topic to my attention. Thank you for showing us polymorphic gates, and I thank you for getting me involved in their research. Without you, this dissertation would not have been possible to complete.

I would like to express my sincere gratitude to my co-authors. Their expertise and collaboration throughout the research process were instrumental in shaping these publications. To Dr. Tian Wang and Dr. Omid Aramoon, for building a solid foundation on top of which I could conduct this dissertation research. Our co-authored articles are not included in this dissertation,

but they are an integral part of my study on polymorphic circuits. To Ms. Zelin Lu and Professor Jinmei Lai for your invaluable contributions to the papers titled 'A Polymorphic Circuit Interoperability Framework' and 'Towards Finding the Sources of Polymorphism in Polymorphic Gates' that are included in this dissertation.

To Devin, my sincerest gratitude for your care and support throughout the years. Your understanding and patience during countless late nights and stressful moments were a constant source of strength. You celebrated my successes, offered a listening ear when doubts arose, and provided a much-needed sense of balance and joy outside of academia. This journey would not have been possible without you.

My deepest appreciation goes to my family, whose unwavering love and belief in me formed the foundation of this accomplishment. To my parents, Juan and Larry, and my sister, Katherine, thank you for always believing in my potential and cheering me on from afar. Your constant encouragement and unwavering support provided the strength and motivation to persevere through challenging times. This dissertation is dedicated to you, with immense gratitude for your love and faith.

Lastly, my journey through academia wouldn't have been the same without the incredible support of my friends. Whether it was sharing a laugh during long nights, offering a much-needed distraction from endless research, or simply reminding me to breathe and keep things in perspective, their presence filled my path with laughter and light. To all my friends, thank you for being my cheerleaders, my sounding boards, and my sanity savers. This accomplishment is as much yours as it is mine, for the joy and friendship you brought made the journey all the more enriching.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1: Introduction

Polymorphism is the quality or state of existing in or assuming different forms. For the purposes of this dissertation, polymorphism is used in the context of electronic gates or circuits, meaning that the circuit can have different functions based on outside conditions. The most common parameter used in this paper for selecting the functionality of the circuit is the supply voltage, $V_{DD}$. The result of this is that when $V_{DD}$ is one value, 3.3V for example, the gate performs one function, such as that of an AND gate, but if that same circuit is operating at $V_{DD}$ = 1.2V, the circuit functions as an OR gate. In this work, many aspects of polymorphic gates, from methods of creation to the applications are examined. This dissertation is structured as follows: Chapter 3 provides an overview of the interoperability of polymorphic gates; Chapter 4 offers an analysis of the quality of the evolutionary algorithm; Chapter 5 introduces time-based polymorphism; and Chapter 6 delves into bringing polymorphism to the subthreshold domain, along with circuit self-testing.

Polymorphic gates are commonly considered to be gates that switch between two functions based on a single parameter, such as $V_{DD}$. However, it is possible for polymorphism to exist across multiple different levels or even across different parameters. For example, it is possible for a circuit to have three or more functions switching with $V_{DD}$: AND at $V_{DD}$ = 1.2, B buffer at $V_{DD}$ = 2.1, and OR at $V_{DD}$ = 3.3. This type of polymorphism is referred to as multi-polymorphism.

Multi-polymorphism also includes polymorphism that uses different parameters, such as both $V_{DD}$ and the circuit sample time: a circuit can be an OR gate unless both conditions are met, $V_{DD} \geq 3.3$ and $T \geq 5ms$. This also introduces the concept of sample rate-based polymorphism, which is referred to as time-based polymorphism.

A time-based polymorphic gate is a polymorphic gate whose function changes based on the time that the circuit is allowed to settle before the result is read. In standard design, it is desired that the circuits to be settled and stable before the value is read, but there are some security applications that can be implemented if the value is only valid for a small segment of time. An example of this type of polymorphism would be a circuit that is an OR gate, but whose 01/10 input values take much longer to settle than the 00/11 inputs. In this case, the circuit would exhibit AND behavior if read with a very quick sample rate and OR behavior if read more slowly. It is important to note that these types of circuits might require a reset between reads to ensure that the value is correct, as multiple inputs of '01' in the previous example might allow the circuit to reach the '1' output even if it is sampled quickly each time.

There are a number of methods used to design polymorphic circuits which are discussed both in this dissertation and in the literature. The two methods primarily used in this work are either through the use of a designer or engineer, hand designing the circuits, or through the use of an evolutionary algorithm. The evolutionary algorithm used here will be detailed later, in section 4.2.1, and is also summarized below.

Evolutionary algorithms are algorithms that allow the program to explore more of the possible solution space than would normally be possible through manual design. In the case of circuit design, the system begins with a set of criteria (such as inputs and expected outputs) and a set of constraints (such as $V_{DD}$ cannot connect directly to ground), and then the system generates an

initial population of random circuits. Each circuit is tested to see how well it matches the criteria and is scored accordingly. The best circuits are selected and changed slightly to create the next generation. As the algorithm progresses, the desire is that it will produce a design that meets all of the desired criteria while following all of the set constraints. Using criteria of different logic functions at different $V_{DD}$ levels allows the evolutionary algorithm to search for polymorphic circuits.

The evolutionary algorithm can output many thousands of gates that meet the constraints set, but these candidate gates may not all be truly polymorphic if they are implemented in silicon. In order to identify a subset of the candidate gates that are truly polymorphic, a further set of tests were developed. These tests are detailed in section 4.2.2 but are intended to help identify any possible simulator or evolutionary algorithm errors or problems.

Polymorphic circuits have many possible applications, including low-power design, circuit self-testing, and hardware security. The low-power and circuit self-testing aspect researched here was through a modification of the evolutionary algorithm to search for polymorphism in the subthreshold, or turn-off, region of transistor operation. Moving polymorphism to subthreshold operation would allow its use in more devices, including super low-power IoT devices, and can tie into the hardware security applications.

Hardware security applications for polymorphic circuits have existed prior to this work, and previous work is detailed in the next chapter. this work was built on in three ways: subthreshold circuit self-checking, subthreshold circuit watermarking, and time-based applications, including embedded signatures, hidden functions, and anti-reverse engineering. A circuit can be self-checking if it is able to, with some sequence of inputs, show different faults on its output(s). A self-checking polymorphic circuit is able to make use of all of the possible functionality within

itself to check the different traces and transistors for faults. The fault model discussed is the stuck-at fault model, where a particular line is stuck at logic high or logic low, either because of a short or open circuit, or because of a transistor fault. Depending on exact gate layouts, different types of fault checking can be implemented.

Time-based and multi-polymorphism can also be applied to hardware security. Embedded signatures, which are signatures that do not affect the ultimate function of the circuit but appear under some set conditions, can be made ephemeral. Hidden functionality can be introduced using any alternative functions of the polymorphic gate, particularly if the gate is multi-polymorphic. Polymorphic gates can also be used to prevent reverse engineering attacks due to their nonstandard layouts, multiple possible functions, and timing needs.

Polymorphic gates change their function based on outside conditions and are particularly tricky to design and simulate. While this is the case, they do have some very interesting applications and have the potential to have a substantial impact on the future of analog and digital circuit design, particularly in the security space.

# Chapter 2: Background and Related Works

Polymorphic gates and circuits were first introduced by Stoia et al [2]. These gates have multiple functionalities that can be expressed based on outside factors, such as temperature or voltage [1]. Figure 2.1 shows one example that can work as an OR, XOR, or AND gate depending on the operating voltage [1]. There has been work on polymorphic gate design in the past. Crha et. al. discussed some of the methods that are used: an experienced designer is referred to as ad-hoc design, evolutionary algorithms, Polymorphic Multiplexing (which is based upon the idea of having separate gates for each polymorphic function and multiplexing their output together), and Polymorphic Binary Decision Diagrams [5]. Of these methods, three are of interest to this project: evolutionary algorithms [3] [1], ad-hoc, and Polymorphic Multiplexing, which is similar in concept to the framework described in chapter 3 [6], specifically the similarity this research and polymorphic multiplexing is separation of the polymorphic element from the rest of the circuit. Other work has created gates that are controlled by external signals, such as the final adder/subtractor presented in [7]. This system is an adder with the external signal '1' and a subtractor when the signal is '0'. Most of the 4 synthesis methodologies and optimizations in the literature tend to focus on larger polymorphic circuits rather than the elementary gates.

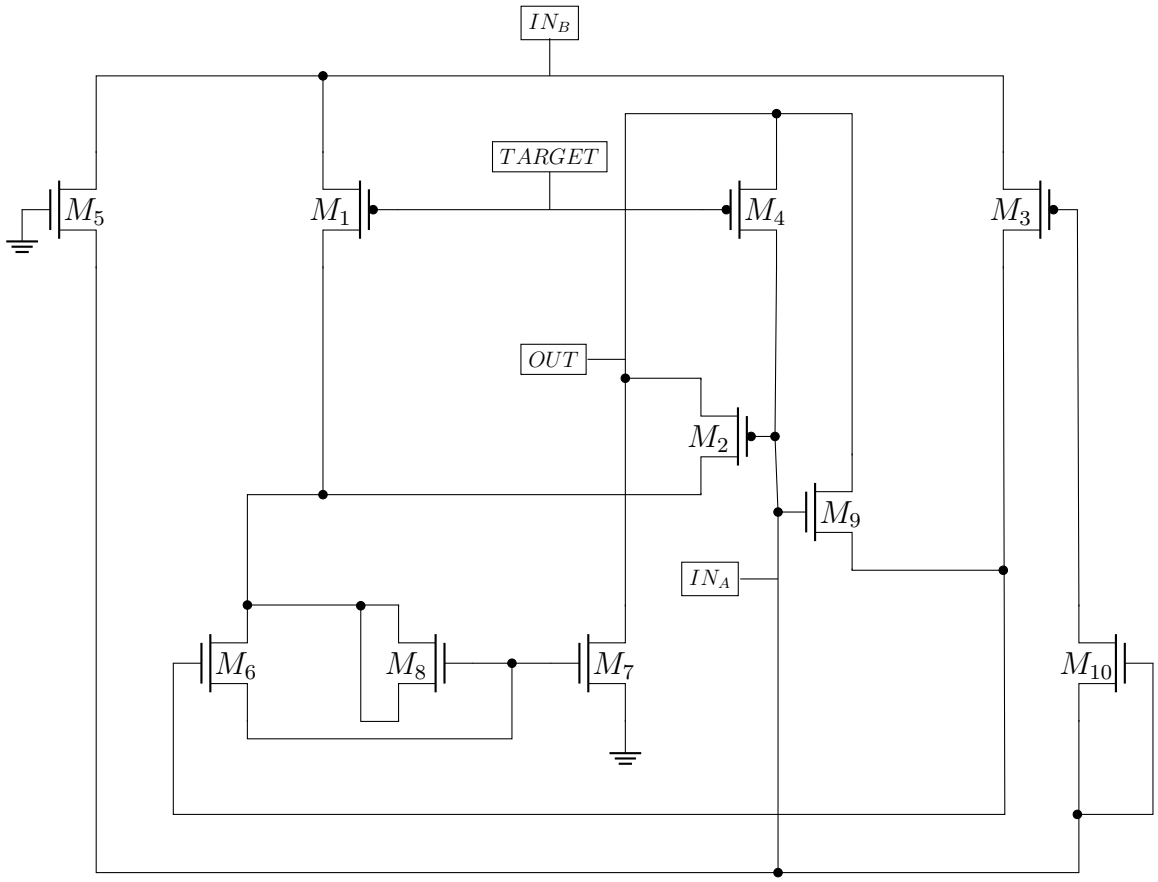



Figure 2.1: This shows the schematic of the redrawn circuit from JPL [1].

## 2.1 Generation Methods

Multiple methods of gate generation are discussed in the literature. The primary method used is the evolutionary algorithm, and information on evolutionary algorithms and the analysis of the resulting circuits is presented below. Other methods of generation include ad-hoc, polymorphic multiplexing, and polymorphic binary-decision-diagrams, as mentioned earlier.

### 2.1.1 Evolutionary Algorithms

Previous evolutionary algorithms are presented by Stoica [3] [1] and Crha [5]. These algorithms function as basic evolutionary algorithms, where a pool of circuits is created, each either

randomly or with a mutation of a circuit in a previous generation, and a fitness function is applied based on how close the circuit is to meeting the desired polymorphic function. Stoica also presents a potential issue to evolutionary algorithms [3]. The issue is that a circuit that is created by an evolutionary algorithm evolved in a particular way, under particular constraints and may not carry into practical application well. If these constraints are not present in fabrication (or if other constraints are present), the circuit may not function once fabricated. One solution described in the paper is to make sure that the fitness function is actually a combination from multiple different simulations of the same circuit (using different conditions). A different issue to evolutionary algorithm is presented in [8] where the fitness function only increments very slowly between generations, this is known as stalling. These will be very useful when the evolutionary algorithm is being run on evolving circuits.

The basic evolutionary algorithm is also shown in the literature [7]. An evolutionary algorithm is the current chosen method because there are not any ready-made design rules to follow for creating polymorphic circuits, and they are discovered mostly through trial and error. This algorithm is seeking to find a polymorphic gate that minimizes the transistor count for given functions that are currently published. The authors are successful in this and are able to design and simulate multiple gates with transistor counts below the values that were published at the time. They used the gates they created to implement a polymorphic adder/subtractor, showing that their designs are valid in application.

A possible modification to evolutionary algorithms is presented in [9]. The paper demonstrates that the standard evolutionary algorithm works well for small circuits, but has an issue when dealing with circuits larger than around 8-15 inputs where the evolutionary algorithm is not efficient and produces unreliable results. It proposes a way to modify the fitness evaluation

step of a standard evolutionary algorithm, which is the bottleneck, by reducing the problem to a satisfiability (SAT) problem and running a standard solver against it. The goal of the proposed algorithm is to reduce the number of gates that are required. The design starts with the creation of a reference circuit using polymorphic multiplexing, which is known to have poor space utilization, and the candidate circuit. These circuits are augmented with a series of XOR gates fed into an OR gate, then the resulting Boolean formula is converted to conjunctive normal form and evaluated with a SAT solver which will evaluate if the two circuits are functionally equivalent. This can be used to create a fitness function that returns the number of gates for circuits that are correct, so the evolutionary algorithm can work on minimizing that value.

One problem with larger evolutionary algorithms is addressed in [8]. The stalling problem is shown in evolutionary algorithms where the fitness of a design does not increase very much with each successive generation. To combat this problem, a few possible solutions are discussed, namely a weighted fitness function and a repair circuit. The weighted fitness function is where certain nodes in an evolutionary circuit are worth more to the fitness function than other nodes. This can be done to prioritize some crucial intermediary nodes over others that may be less important. This option was able to take a 6 input, 6 output circuit and reduce the number of generations from 10,244,650 generations to 2,514,043 generations. In order to create a system that would reduce it even further, the evolutionary repair circuit was designed. In the repair algorithm, once an evolutionary algorithm stall was detected, a repair circuit would be generated. This repair circuit would "fix" the circuit by adding components that would enable the correct output to be generated. Once the repair circuit was in place, the evolutionary algorithm would resume and attempt to minimize the area of the total circuit. This would cause the fitness function to jump dramatically and remove the stall from the system.

### 2.1.2 Analysis of Evolved Circuits

Compared to the design and applications of polymorphic gates, there is little effort reported on the analysis of the polymorphic gates evolved by evolutionary algorithms. Existing research primarily focuses on improving the evolutionary algorithms to increase their robustness and avoid the limitations mentioned earlier. During this literature review, there is no reported analysis of the structure of the circuits obtained from the evolutionary algorithms beyond the functionality of these polymorphic gates being validated by Spice simulation.

The techniques and experiences shared in [3] are valuable during the silicon implementation of the evolved polymorphic circuits, but they did not offer much insight into the sources of polymorphism. The approaches used to improve the efficiency of evolutionary algorithms are rather standard techniques such as fine-tuning the fitness function, the parameters, mutation strategies, and the evolution process. There is no systematic study from circuit analysis' standpoint to assist the evolutionary algorithms to converge or the evolved gates to be more robust. In this paper, a more detailed look at the behavior of these circuits and the structures that may evolve in these algorithms is taken. This analysis aims to identify some sources of polymorphism in these circuits and provide insights that can help improve their design and performance, but is not intended to be a complete explanation of all possible sources of polymorphism from evolutionary algorithms.

### 2.1.3 Other Generation Techniques

An alternative to generating large polymorphic circuits using evolutionary algorithms or polymorphic multiplexing is presented in [11]. The problem presented by Fiser and Simekis that,

| $K_3$ | $K_2$ | $K_1$ | $K_0$ | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | $A\bar{B}$ |
| 0 | 0 | 1 | 0 | $AB$ |
| 0 | 0 | 1 | 1 | $A$ |
| 0 | 1 | 0 | 0 | $\bar{A}\bar{B}$ |
| 0 | 1 | 0 | 1 | $\bar{B}$ |
| 0 | 1 | 1 | 0 | A⊙B |
| 0 | 1 | 1 | 1 | $A+\bar{B}$ |
| 1 | 0 | 0 | 0 | $\bar{A}B$ |
| 1 | 0 | 0 | 1 | $A \oplus B$ |
| 1 | 0 | 1 | 0 | $B$ |
| 1 | 0 | 1 | 1 | $A+B$ |
| 1 | 1 | 0 | 0 | $\bar{A}$ |
| 1 | 1 | 0 | 1 | $\bar{A}+\bar{B}$ |
| 1 | 1 | 1 | 0 | $\bar{A}+B$ |
| 1 | 1 | 1 | 1 | 1 |

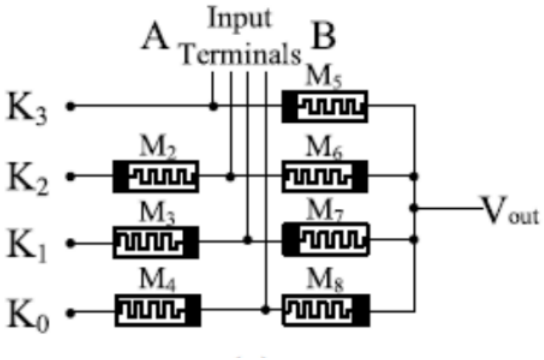



Figure 2.2: Resistive RAM Polymorphic Gate, dependent on Kx inputs to select gate, with truth table [10]

while evolutionary algorithms generate a solution, there is no way to know if it is the optimal solution. The proposed solution implements the polymorphic circuit as a Boolean Satisfiability Problem (SAT) and attempts to solve the problem with 'n' nodes. If the problem is unsatisfiable, then no solution with 'n' nodes exists, so 'n+1' nodes are used and the process is repeated. Polymorphism is implemented in the SAT graph by using "polymorphic edges" which are edges that will negate the output of a node under the polymorphic condition. These polymorphic edges are implemented in design using currently existing, minimized, polymorphic gates. This method is useful for generating optimal circuits from currently existing polymorphic gates rather than generating new polymorphic gates.

Another paper shows a possible avenue of polymorphic gate generation with a Resistive RAM (RRAM) implementation [10]. In this paper, a RRAM structure is created that takes in 4 inputs (K) which are used to select the gate desired in the RRAM structure. The structure is shown in figure 2.2, with the truth table. The polymorphism in this structure is due to the changing K inputs, which makes this an external input polymorphic gate. The gate is first initialized by

grounding the K inputs and the $V_{OUT}$ terminal, with the input terminals set to a high state. Then the logic inputs A and B are set by putting these on the input, raising the K inputs high for A, or setting $V_{OUT}$ high for B. Once the logic inputs are set, the gate function is selected with the K inputs. This circuit implements all 16 possible 2 input logic functions, and therefore is resistant to a reverse engineering attack, since the target function is completely obfuscated without the K input key. Polymorphic multiplexing and PolyBDD (Polymorphic Binary Decision Diagrams) are demonstrated in [12]. Polymorphic multiplexing is shown to be a straightforward method of generating polymorphic circuits, where each of the desired functions are implemented using a standard CMOS process and the results are switched between depending on a polymorphic multiplexer. The downside to this methodology is that this uses a substantial number of transistors and power, since all possible results are being generated, but all but one of these results are being ignored by the multiplexer. In an effort to reduce the use of power and space, Polymorphic Binary Decision Diagrams (PolyBDD) was proposed. In PolyBDD, a binary decision diagram is modified to allow for integers on the leaf nodes: 0-15, representing the 16 possible logic functions involving two inputs. The end result is a BDD that can be optimized using standard optimization techniques. During synthesis, the only difference is that the final leaf nodes are implemented as elementary polymorphic gates that already exist.

## 2.2 Applications

A number of possible applications exist for polymorphic circuits. Some that have been explored are physically unclonable circuits [13], low-power design [14], and intellectual property protection (using either circuit watermarking or circuit obfuscation or self-checking circuits [15]).

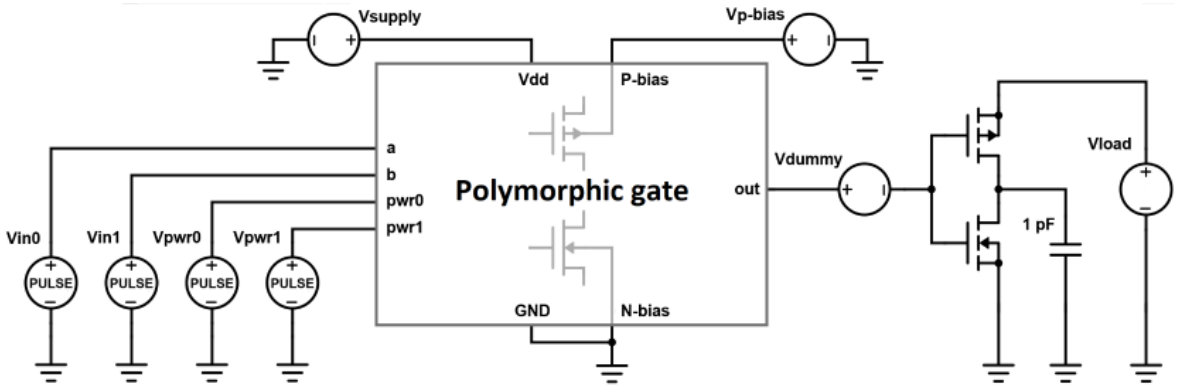



Figure 2.3: Power consumption analysis hardware added on for correct analysis in HSPICE [14]

Other potential applications include data encryption and random number generation.

### 2.2.1 Low Power

Low power design is evaluated in [14]. Currently low power design is also a target of security primitives such as signature aggregation and encryption [16]. One possible low-power design is based on the polarity of the supply rails. These gates are taken from the PoLibSi library, which has numerous gates of varying levels of quality. This allows the checking of a multitude of gates to evaluate the best for low power design. In order to accomplish the power analysis in HSPICE additional circuit structures, shown in figure 2.3, were added. The reason for this is that HSPICE power estimation was found to be wildly inaccurate for these devices. The final results of the analysis revealed that size optimized polymorphic structures in the PoLibSi library tended to be more power inefficient than those that used slightly more transistors.

### 2.2.2 IP Protection and Circuit Self-Checking

Fault detection and self-checking are problems that designers face [17]. Currently self-checking and IP protection applications exist for polymorphic circuits. Self-checking applica-

tions described in the literature involve creating an adder with a $V_{DD}$ controlled polymorphic NAND/NOR gates that, if no error is present, remain steady even across the $V_{DD}$ change point. This proposed adder can be used to detect stuck-at-fault errors, but the idea of a polymorphic self-checking circuit can be expanded to include any type of fault [15]. In the current literature, IP Protection using polymorphic circuits falls into two areas: circuit obfuscation and circuit watermarking. Circuit obfuscation, described by McDonald et. al [18], is where the functionality of the circuit is hidden by the polymorphic key. The circuit must be run with the correct operating conditions ($V_{DD}$, temperature, etc..) in order for the circuit functionality to be seen. This type of IP protection is to prevent a malicious reverse engineer or a malicious fabrication facility from having easy access to the circuit functionality. Circuit watermarking is also described by McDonald et. al. [18] and involves replacing a subset of gates with polymorphic ones such that any other group that copied the design would also include the exact same gates, thus proving that the circuit design was stolen. Additionally, Koteshwara et. al. describes a method where the circuit is obfuscated with a changing key [19]. This method could be adapted to a polymorphic system by having the changing key be changing environment variables instead. Self-checking and IP protection using polymorphic gates has been described in the literature, and can be done with a reasonable amount of certainty.

### 2.2.3 Physically Unclonable Circuit

A PUF is a device that is designed in such a way that each individual unit will have a different response to a given challenge, so that every time the device is made, they are physically different (and thus unclonable). PUFs have been discussed at length in the existing literature. Lu

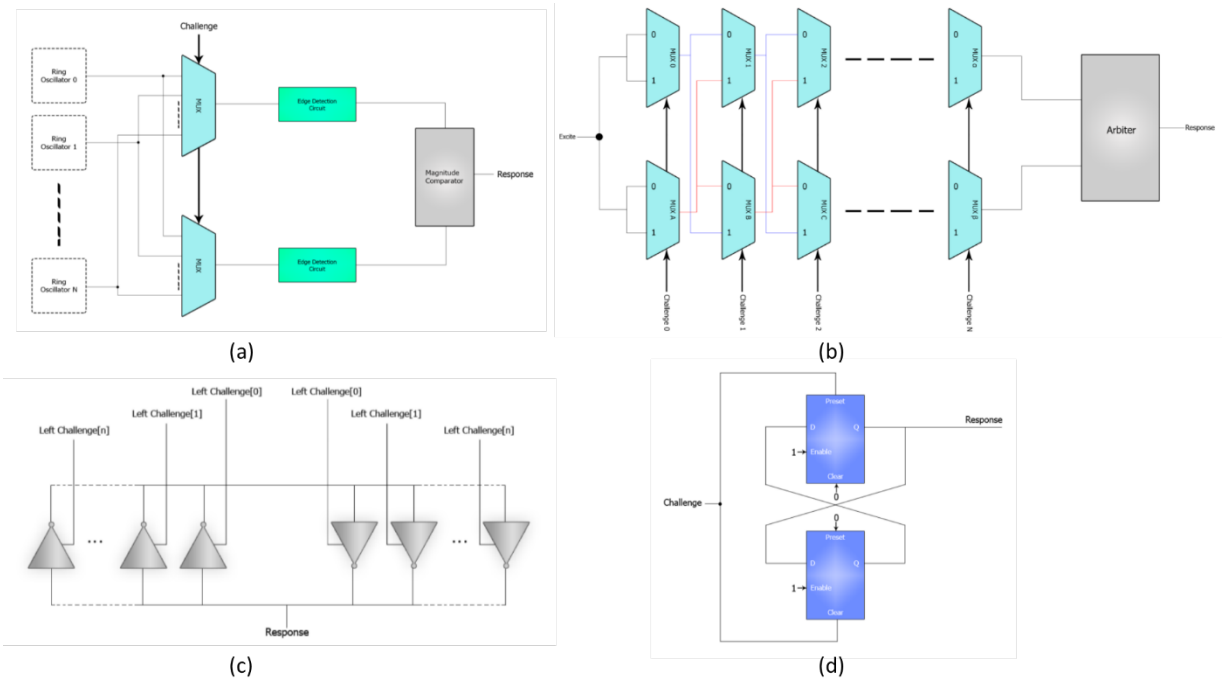



Figure 2.4: a) Ring Oscillator PUF, b) Arbiter PUF, c) SRAM PUF, d) Butterfly PUF [20]

et. al. [21] and Mispan et. al. [22] each discuss low power PUFs, but Konigsmark et. al. [13] discusses a polymorphic PUF. In that work, a PUF is presented that the response is decoupled from the challenge in such a way that a modeling attack cannot be effective. The attack described uses machine learning/artificial neural networks to model the PUF's state. In this case, the polymorphism is due to the random changing of what the response is, but this can in theory be adapted to work with polymorphic circuits, as discussed in future work below.

The different types of PUFs are described in [20]. The two main types of PUF are memory based and delay based PUFs. A memory based PUF is a PUF that relies on the memory state tending to a particular value during power up, either by using SRAM in an SRAM PUF or by using latches as is done with a butterfly PUF. Delay based PUFs are PUFs that rely on two identical components operating at a slightly different speed, thus resulting in a different delay. Examples of a delay based PUF are ring oscillator PUFs, where the speed of different ring os-

cillators are compared and the result is used to generate a response, and arbiter PUFs, where the fastest component is used to generate a response.

Figures for these different types of PUFs are given in figure 2.4. The common attacks against PUFs are also presented in [20]. The attacks given can be broken down into four groups: modeling, side channel, fault injection, and invasive. Modeling is where the internals of a PUF can be recreated using a small number of challenge-response pairs to extrapolate the remainder of the set. Side channel attacks use external information, such as power consumption, to extract secret information from the internals of the PUF. Fault injection is where an adversary will operate the PUF in some extreme condition, such as high temperature or voltage, to force the PUF to operate outside of its normal state, which can overwhelm the fault correction circuitry and reveal information about the internals of the PUF. An Invasive attack is where an attacker will reverse engineer the device, usually by disassembling the device and analyzing the internals very carefully.

There are defenses to each of these attacks, ranging from additional obfuscation hardware and large challenge-response sets to help defeat modeling, fault injection, and side channel attacks to tamper evident design that changes the responses once the device has been opened. The attack methods used are improving and may render some of these defenses unusable in the future.

A modeling attack, like the one described earlier, is given in [23]. This paper details an attack that uses an evolutionary algorithm to build a model of the internals of a multiplexing PUF. The PUF described is quite simple and shown in figure 2.5. In this PUF, 128 ring oscillators, each comprised of three inverters chained together, are connected to a multiplexer whose select bits comprise the challenge. This structure is duplicated (so each side of the PUF has 128

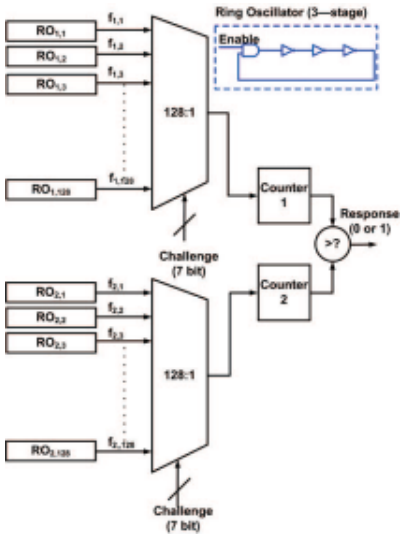



Figure 2.5: Ring Oscillator PUF, modeled with machine learning to demonstrate modeling attack [23]

oscillators and a multiplexer) and the challenge is given to both. The outputs of the multiplexers are given to a counter, whose output is then given to a comparator. The output of the comparator is the response. Essentially, this will select one ring oscillator from each side, depending on the challenge, and will respond based on which one has the highest clock speed. The evolutionary algorithm seeks to model this PUF by creating a tree diagram whose nodes are functions (AND, OR, or NOT) and the leaf values are bits of the challenge. This evolutionary algorithm was run and eventually outputs a response that can be around 84% accurate (best case presented) using only 25% of the possible responses. This algorithm was run over 10 generations, and it is possible that running for longer could yield better results. Some other aspects of machine-learning algorithms with respect to PUFs are given in [24]. This paper discusses Probably Approximately Correct Learning, in which a system such as a PUF is modeled pursuant to a particular level of accuracy and defined confidence interval. Two key points are discussed when designing a machine learning model to attack a circuit: the circuit must be able to be somewhat accurately modeled by the machine, and the machine must be searching for results with a predefined confidence interval

and accuracy. This paper uses XOR arbiter PUFs, which are chains of arbiter PUFs chained together with XOR gates, as the target of the analysis. These particular PUFs can be modeled fairly accurately using linear threshold functions (LTFs), and thus satisfy the first requirement. Other PUF designs mentioned, such as the bistable ring PUF, are not accurately modeled mathematically, and thus cannot reach 100% accuracy even with a complete set of challenge-response pairs, instead capping out at around 95%. This ties into the second requirement of looking for a predefined target accuracy and confidence interval: there must be a set target with correct parameters, otherwise the true security of the system cannot be measured.

## 2.3 Subthreshold Design

Subthreshold circuits were initially presented in 1972, with many applications being realized in the 1990s [25]. The primary motivation for subthreshold circuits was at the time, and still is, to reduce power consumption [25–27]. Subthreshold circuits allow for much lower power by operating using the cutoff region of a transistor, below $V_{TH}$.

### 2.3.1 Subthreshold Operation

Subthreshold circuits operate when the voltage used to switch a transistor is lower than the $V_{TH}$ of the transistor itself. This region, in super-threshold circuits, is considered ”off”, but some current does flow. This current flow is much less than super-threshold operation current, and a logarithmic curve that illustrates this is shown in 2.6.

Subthreshold circuits do, however, trade low power consumption for speed. The speed at which a subthreshold circuit can switch is substantially slower than that of a super-threshold

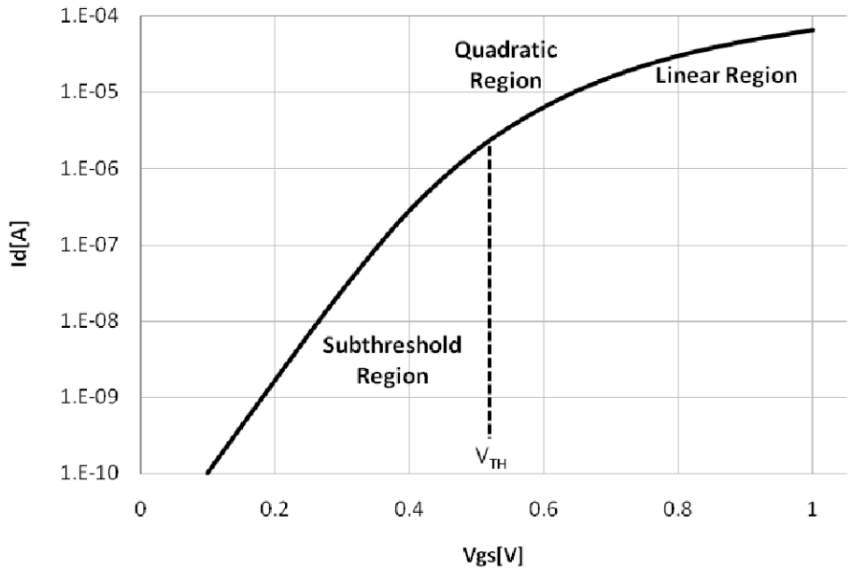



Figure 2.6: This figure shows the IV curve of a MOSFET transistor in both above and subthreshold operation [28].

design [25,29]. This trade-off is one that must be considered when selecting subthreshold designs for use and eventual fabrication.

Subthreshold circuits also fall victim to reliability degradation. Because of the mode of operation for subthreshold circuits and current fabrication processes, the voltage range that will allow for correct operation can be very precise, which can lead to some circuits being unreliable [30, 31]. To address this unreliability, fault detection is required. Fault detection in the subthreshold domain is presented in [30] for use in simulated circuits.

### 2.3.2 Subthreshold Applications

There are many applications for subthreshold circuits. Subthreshold circuits are extremely well suited to applications that are either low power, low speed or applications that might only require high performance in a burst mode [26]. Subthreshold circuits, in a burst mode situation, would operate with the use of multiple $V_{DD}$ rails and switch between super and subthreshold

depending on performance requirements [25]. Other applications of subthreshold designs include: implantable/wearable devices [32], hardware primitives/PUFs [33], and random number generators (RNGs) [34].

Implantable devices are devices such as an EKG or pacemaker that do have particular power and security constraints, and often use techniques such as energy harvesting to maintain power [29, 32, 35]. These devices must use extremely small amounts of electricity, but must also be secured from attacks on a foundational level, since replacement can be difficult and they can have deadly consequences. Due to this, subthreshold circuits must be created that meet both the low power and the security requirements [35].

Similar requirements also exist for wearable devices in that they need power harvesting, security, and power efficiency [32]. These devices are more accessible than implantable devices, but must also be secured in order to protect personal information and safety. Each device also must be power efficient enough to last throughout the day or use-cycle of the device without inconveniencing the user.

Subthreshold devices can also be used as hardware security primitives beyond securing the user. One possible design is in super low power wireless communications, such as IoT devices, and the error correcting codes used within that ecosystem. In the particular situation described in [26], a polar code is demonstrated. This code allows for a substantial decrease in power while maintaining a high level of error correction. Other primitives can also include PUFs. In [33], a memrister is used in subthreshold as a PUF by sweeping different currents and noting the voltages seen. A random number generator is demonstrated in [34]. This RNG is implemented in subthreshold and has a high entropy as measured by NIST standards, with a throughput of 400kbps with a $V_{DD}$ of 0.3V. This example also demonstrates the speed difference with subthreshold. A

paper demonstrating a random number generator published around the same time presents speeds over 300Mbps [36], a substantial improvement in speed at the cost of power consumption.

Subthreshold design has, to this point, not been shown to behave in a polymorphic way. If polymorphism can be shown to apply to the subthreshold domain, then polymorphic applications that exist in super-threshold spaces can be brought to the subthreshold domain. As stated earlier in this chapter, many applications exist in the super-threshold space involving polymorphism, including multiple, such as PUFs and error correction, could be used in the subthreshold domain where solutions are still being sought.

# Chapter 3: Polymorphic Circuit Interoperability Framework[1]

## 3.1 Chapter Overview

Polymorphic gates are logic gates that change their function based on some external factor. Since the 1970s, polymorphic circuits have been theorized, created, and their effects studied. Some possible polymorphic gates are gates that change their function based on voltage (e.g. a gate that is a NAND gate when $V_{DD}$ is 1V and NOR at 3V), temperature, or an external signal [3,4,7]. Historically, these gates have been difficult to design, with one common design methodology being evolutionary algorithms [1,5]. These algorithms attempt to design a full polymorphic gate, such as the NAND/NOR gate mentioned above.

This paper proposes the Polymorphic Interoperability Framework for the design and construction of polymorphic gates. This framework separates the creation of the polymorphic functionality, such as switching due to the supplied voltage, from the gate functionality, such as NAND/NOR. This paper also presents a method for the reconfiguration of currently existing polymorphic gates into this framework.

[1] From paper [6]

## 3.2 Polymorphic Framework Overview

The Polymorphic Interoperability Framework contains two main segments: the polymorphic element and the switchable gate. These two segments are combined together with an interface consisting of a series of signals and at least one signal is required.

### 3.2.1 Polymorphic Element

The polymorphic element exists as the portion of the system that is acted on by the environment to change state. This change is output to the switchable gate through the interface. A polymorphic element that switches state based on the voltage of the system is shown in figure 3.1. This element is using a single output through the interface. It is possible to use multiple outputs through the interface, utilizing either multiple levels of the same polymorphic element, or combining multiple polymorphic elements into a single group.

The polymorphic element is the crux of the framework. Currently existing methods can be employed to create this polymorphic element [5]. Of these methods, two will be highlighted here: an evolutionary algorithm and a skilled designer.

One of the most common methods of polymorphic circuit design is to use evolutionary algorithms [3]. These algorithms start by randomly combining transistors, testing them, and combining different partial solutions to eventually create a circuit that completes the needed functions [4]. In order to use an evolutionary algorithm with this framework, the target design must be set to interact with the interface, which is commonly just a single signal or a small number of signals, depending on the type of switchable gate, as detailed below. When compared to a standard polymorphic gate, this requirement makes for a notably simpler method for design

since only a single output condition must be created instead of every output of a gate's truth table. This also highlights one of the major benefits of this framework: existing circuit reuse.

Much work has been done towards developing different polymorphic circuits [1, 3, 5]. To leverage as much previous work as possible, it was important for this framework to be able to make use of currently existing polymorphic circuits. In order to convert a currently existing polymorphic gate to the framework, an input combination that expresses a different output must be forced. For example, in [4] a NAND/NOR temperature polymorphic gate is displayed. If input A is forced high and B is forced low, then the output switches exactly with the polymorphic state. This concept can be extrapolated to a generic gate because by definition a polymorphic gate must be a combination of two distinct gates, and by definition of distinct gates there must exist some input combination that causes the output to differ. Using this principle, polymorphic elements can be crafted from existing gates described in the literature.

Another option for creation of these polymorphic elements is for a skilled designer to invent them [5]. Designing the polymorphic element is an easier task when compared to designing the entire gate since the design must only conform to the interface, or a very sparse set of signals, and not handle the inputs that a gate would. Two of the polymorphic elements described and simulated in the section below have been designed by hand.

### 3.2.2 Switchable Gates

Switchable gates are gates whose function depends on the input from external signals. The external signal, in this case, comes from the polymorphic element through the interface. Examples of switchable gates can be seen in any previous work that depends on an external

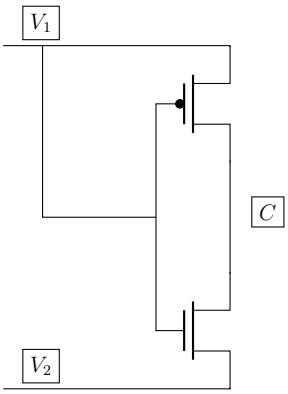



Figure 3.1: Example showing a polymorphic element where 'C' depends only on the voltage difference between the top and bottom rails, $V_1$ and $V_2$.

Table 3.1: Truth Table for NAND/NOR gate

| **A** | **B** | **C** | **Out** |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

signal [1], and these circuits can, in general, be used interchangeably with the switchable gates presented here.

In this chapter, one switchable NAND/NOR gate is discussed. This circuit was created with standard CMOS creation methods based on the truth table in table 3.1, with C being the control signal from the polymorphic element. The resulting circuit is shown in figure 3.2. A particular benefit of this framework is that the switchable gates are able to be minimized using standard circuit minimization techniques, such as using a karnaugh map.

## 3.3 Circuit Examples and Simulation Results

A polymorphic element designed to change result based on clock speed is shown in figure 3.3. The gate on M3 is constantly held at .7V, which allows for relatively small amounts of current to be pulled to ground from the clock signal. This forces the value of the clock to remain close to zero for high clock speeds because it cannot charge the capacitor in the time that the clock signal is high. This forces the output signal (C) to be high, which goes to the interface and the switchable gate.

A polymorphic element created for voltage-level based switching is shown in figure 3.3. This circuit has varied W/L values for the M3 transistor, which causes the output to switch around 1.2V, as shown in figure 3.4. This switching voltage becomes the threshold for the output changing from 1 to 0 and is the signal that is given to the interface. The switchable gate designed for testing these polymorphic elements is a NAND/NOR gate for the control signal being 0/1, respectively. This circuit becomes a majority wins circuit where if two or more low signals are detected, the circuit output is high. The circuit used is shown in figure 3.2. When 'C' is tied to the interface, the circuit behaves as expected, as shown in figure 3.5.

The polymorphic elements are connected to the switchable gate via the interface (signal 'C') and simulated across all A/B input possibilities and the polymorphic parameter. The results were as expected and documented in table 1 as well as shown in figure 3.6. In figure 3.6, A and B show the result of the voltage polymorphic element while C and D show the result of the clock speed polymorphic element. The voltage simulation was run at 5 volts and 1 volt for A and B respectively. The clock speed was run at 50 MHz and 50Hz, respectively. The signal should be sampled on the rising edge of the clock.

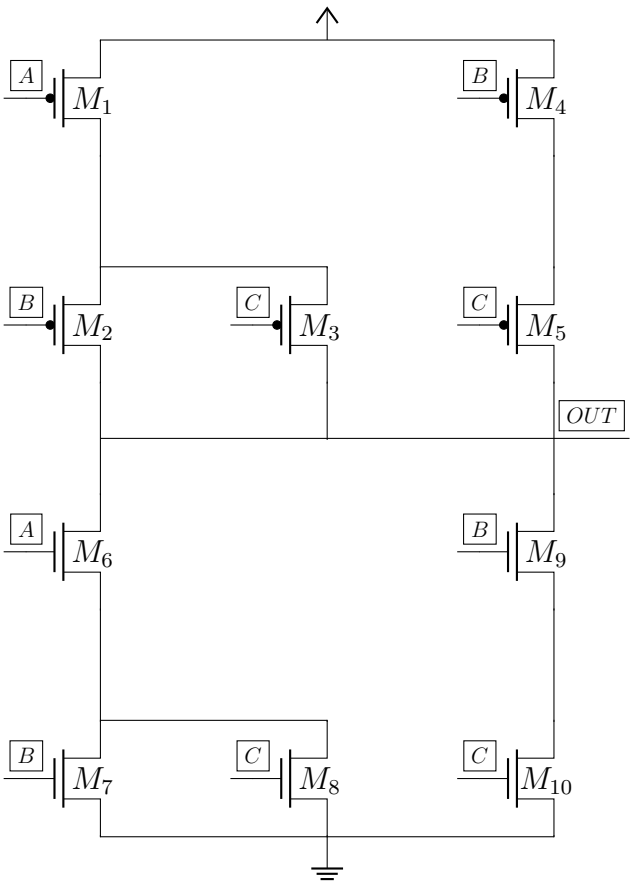



Figure 3.2: Switchable Gate to implement NAND/NOR gates. Switches from NAND to NOR based on 'C' going from low to high.

## 3.4 Potential Applications

### 3.4.1 Digital Watermarking

Multiple factors are critical to the design of a functional digital watermark: size and speed of the implementation, difficulty of detection of the watermark by potential intellectual property thieves, and uniqueness of the watermark possibilities [37]. The polymorphic framework can be used to design a watermark that can meet each of these requirements in a simple way.

This framework can be used in such a way that, during normal operation (with no watermark displayed), there is no discernible change in speed. This is achievable because the frame-

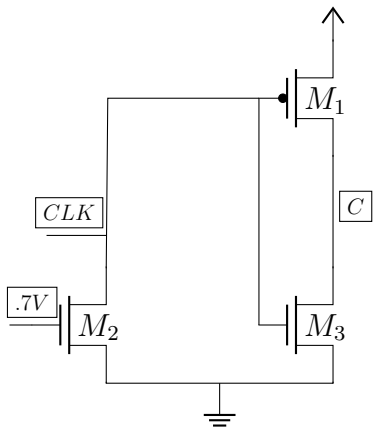



Figure 3.3: This figure shows a polymorphic element that switches output based on the speed of the CLOCK signal switching.

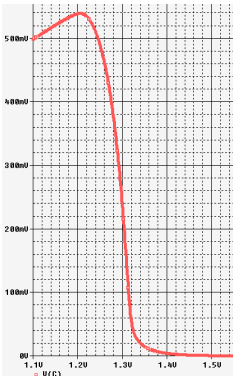


Figure 3.4: This graph shows the value of 'C' as $V_{DD}$ changes for the circuit in figure 3.1.

work only changes the circuit operation during certain operating conditions. We present a circuit that only adds a single gate delay to the input of an adder that can be used to force the adder to a particular value chosen by the designers. Figure 3.7 shows the design of such a circuit that uses an outside polymorphic element (coming in to the signal labeled P). As illustrated in the figure, the inputs A and B are added with an adding circuit but each input signal has a single gate in its path. All signals in A have an AND gate (ANDed with P) and all signals in B have either an AND gate or an OR gate (with P and PBar, respectively). The ultimate result forces the output to be either a 0 in the case of A or a 1 in the case of B. Figure 3.8 shows these gates, along with the truth table associated with the P signal from the polymorphic element. Using this ability to force

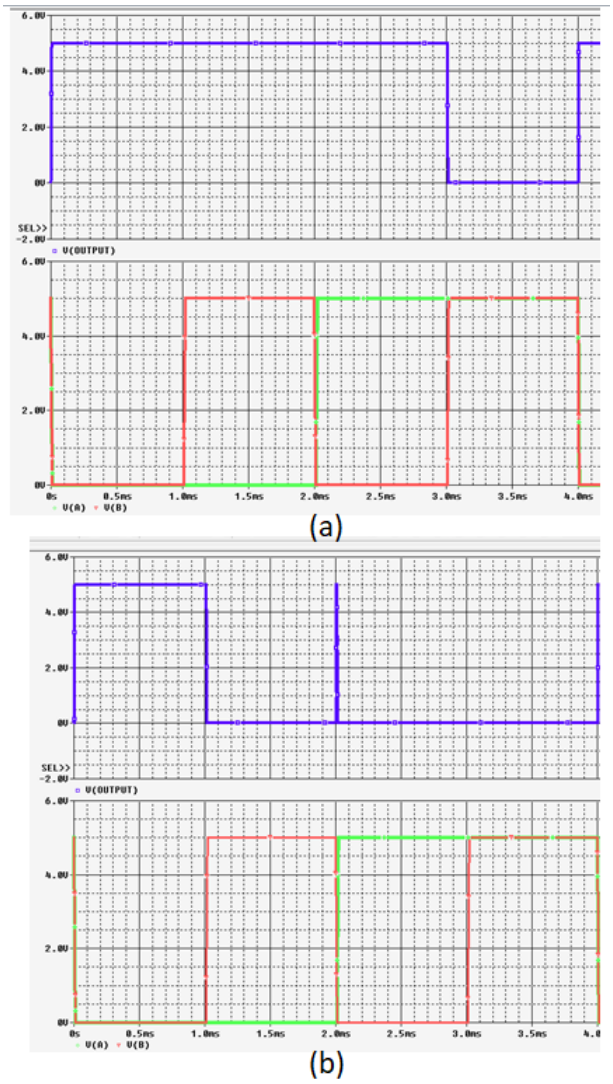



Figure 3.5: This figure shows the output of the switchable NAND/NOR gate. The first chart shows with 'C' low, the second is with 'C' high.

the input of A to 0 and B to whatever bit pattern is desired, the circuit can be watermarked.

This design does have a flaw with respect to obscuring the watermark. If a malicious actor were to look at this design, they would be able to determine that something unexpected was incorporated in the design, thus revealing the presence of the watermark. One possible method of addressing this is to utilize multiple polymorphic elements in such a way that the attacker would need to brute force the design to discover the actual functionality of the watermark

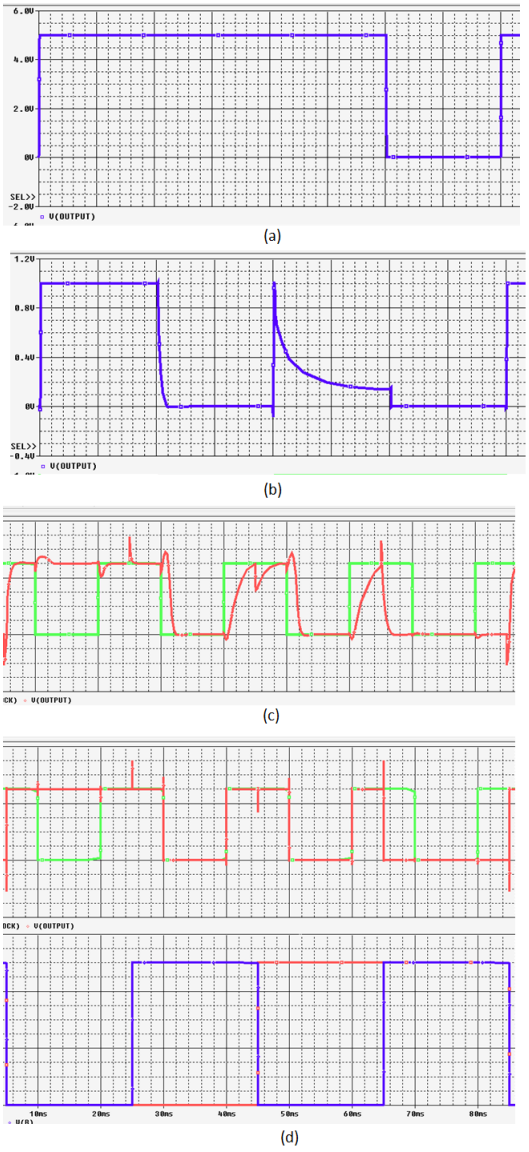



Figure 3.6: A and B show the result of the voltage dependent polymorphic element tied to the switchable gate. C and D show the result of the clock speed dependent polymorphic element

section. Another method is to watermark the circuit in an area where switching gates are already implemented. Utilizing a section with existing switching gates would allow for the designer to completely hide the presence of the digital watermark section of the circuit behind the functions of the gates (hiding the watermark in unused states of the system). This can be done by ensuring that the polymorphic states are used in other sections of the system. A possible implementation of this is shown in figure 3.9. In this figure, the A input has a similar configuration to before,

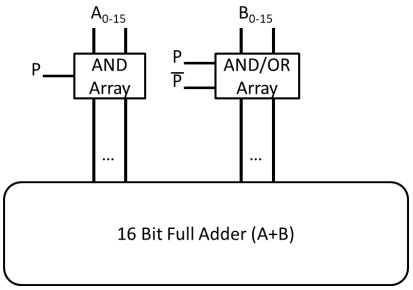
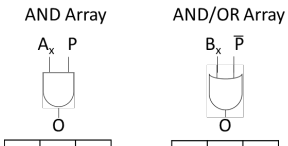



Figure 3.7: This figure shows an adder with an externally controlled polymorphic signal, P, used to show a watermark at a particular value set by the designer.




| P | $A_x$ | O |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| P | $B_x$ | O |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Figure 3.8: This figure shows the truth table for each input signal to the adder in figure 3.7.

Table 3.2: Function Table for Polymorphic Adder

| $P_0$ | $P_1$ | State |
|---|---|---|
| 0 | 0 | Watermark |
| 0 | 1 | Subtract |
| 1 | 0 | Add |
| 1 | 1 | Indeterminant |

except it is ANDed with the XOR of two polymorphic elements. The bits from input B, however, are put through a much more complex polymorphic structure: a switchable NAND/OR gate that is triggered by the $P_0$ signal and operates on Bx and P1. This is shown in figure 3.10. The resulting system state controlled by the $P_0$ and $P_1$ signals is shown in table 3.2. Please note that to correctly place this system into subtraction, the least significant bit of input B needs to go through the AND array instead of the NAND/OR Gate Array. This enables the system to use two's compliment subtraction.

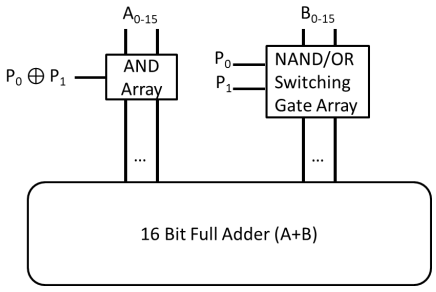
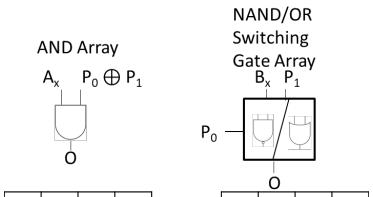



Figure 3.9: This figure shows another polymorphic adder. This adder changes function based on the inputs $P_0$ and $P_1$ based on the functions in table 3.2




| $P_0$ | $P_1$ | $A_x$ | O |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

| $P_0$ | $P_1$ | $B_x$ | O |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Figure 3.10: This figure shows the truth table for the inputs to the adder in figure 3.9.

### 3.4.2 Circuit Obfuscation

One method of circuit obfuscation is done by implementing multiple aspects of the circuit with switchable gates. If this is done, then the circuit must be run with the correct operating conditions (therefore the polymorphic elements are all in the correct state) in order to "unlock" the functionality of the circuit. One possible example of this would be to change some inverters to an XOR gate. An XOR gate, acting as a switchable gate, has the effect of either passing the original signal through or inverting it, depending on the signal from the polymorphic element. If some of these gates are also included in the general path of the circuit, you can get the effect of

circuit obfuscation without the need for a large number of additional transistors. This design can be made exponentially more difficult to break by using multiple different types of polymorphic elements, thus "encrypting" the circuit with a longer key.

This method has a shortcoming in trying to truly secure a design: an attacker can see that there is a circuit there, and they can brute force the "key". This can be addressed with a method unique to these switchable gates: multiple valid circuits. In figure 3.9 earlier we presented a circuit that has switching gates on some of the inputs. This allows the design to either add A and B or subtract B from A. The obfuscation presented here is a simple example, but shows that it is feasible to implement a polymorphic design that has multiple valid circuit configurations. This means that an attacker would not necessarily know which configuration is needed to operate the overall design correctly.

Another method of circuit obfuscation is dynamic functional obfuscation [19]. In this obfuscation scheme, the valid key changes with time. If a similar scheme were used with a polymorphic element as the key, one would have to vary the operating conditions during operation to retain correct functionality.

### 3.4.3 Physically Unclonable Function

A Physically Unclonable Function (PUF) is a circuit that has physical characteristics that vary due to the manufacturing process and can use these variances to generate challenge-response pairs unique to that particular device [22]. In addition to this property, PUFs also ideally have uniqueness, repeatability of the desired output, are inherently generates, and are unclonable even under the same manufacturing process [21]. Polymorphic elements and switchable gates can be

incorporated into this design as a simple addition to currently existing PUFs. These polymorphic elements can add an additional piece to the challenge: the current operating conditions.

## 3.5 Discussion

A framework for the creation and modification of polymorphic circuits was introduced. This framework was built around the idea of separating the functional gates from their polymorphic components. Once these elements have been separated, they can be interchanged at will to create new polymorphic gates, allowing optimizations created for one gate to be applied across numerous different polymorphic gates. Examples given are based around a NAND/NOR switchable gate and multiple polymorphic elements: voltage switching, clock speed switching, and temperature. Simulation results of this process show expected results that confirm the polymorphic behavior of this framework. Possible applications of this framework are also presented and detailed, including circuit watermarking, circuit obfuscation, and physically unclonable functions.

# Chapter 4: Finding the Sources of Polymorphism in Polymorphic Gates[1]

## 4.1 Chapter Overview

The earliest polymorphic gates were discovered by NASA's Jet Propulsion Laboratory (JPL) [1]. These gates were found using specially crafted evolutionary algorithms and the extensive use of Spice simulations. This methodology has since become widely used for creating polymorphic gates due to its effectiveness [1, 2, 4, 5]. These approaches, however, are not without limitations. One such limitation is the generation of gates that are highly optimized for the specific fitness function employed by the evolutionary algorithm. These gates may not generalize well to other simulation settings or real-world fabrication processes. Additionally, evolutionary algorithms themselves are susceptible to stalling, potentially hindering the identification of optimal solutions [3]. Also, the success of finding one polymorphic gate offers little insight toward finding other polymorphic gates and both the evolutionary algorithm and the Spice simulation can be time-consuming. Despite the promising advantages of polymorphic gates in area efficiency and low power design [38] as well as hardware security [39–44], the lack of effective methods in generating polymorphic gates becomes the bottleneck of commercial adoption of such technology.

The potential of using various technologies, including polymorphic gates, as hardware se-

[1] From paper [49]

curity primitives is discussed in [39]. Specific circuit watermarking [4,40] and fingerprinting [41] techniques are proposed across the literature. An asynchronous circuit design is reported to implement dual-cryptographic regimes in one polymorphic circuit controlled by supply voltage [42]. The concept of partial polymorphic gates is proposed in [43] and used for circuit authentication. Most recently, a polymorphic gate-based watermarking method that is robust against removal attacks was reported [44]. One common theme behind these approaches is that select standard library cells will be replaced with polymorphic gates so the circuit performs the required functionality under normal conditions. The security features, such as a circuit watermark, fingerprint, or circuit identification tag, are implemented with polymorphism when the operating condition is changed. This makes polymorphic gates an ideal candidate for the protection of design intellectual property (IP) against piracy [45]. However, because these gates make use of the small group of known polymorphic gates, these systems can be weaker than the theoretical design.

This research is motivated by the need to better understand polymorphic circuits and the possible sources of polymorphism. This understanding can ultimately allow designers to tune and craft polymorphic circuits more precisely and efficiently for a wider range of applications. For this purpose, an in-depth analysis on the polymorphic circuits that were discover was performed, as well as an analysis of others that are reported in the literature. Interesting and promising results from this analysis were obtained. More specifically, the following contributions are made in this chapter:

- A methodology to identify polymorphic circuits that are more likely to reveal sources of polymorphism is proposed.
- These circuits are analyzed to explain their polymorphism and discover several structures

in these evolved circuits that are different from traditional CMOS circuits.

- Two polymorphic gates were created using some of the structures discovered, without the use of an evolutionary algorithm.

The rest of the chapter is organized as follows: In Section 4.2, the methodology to choose the most representative polymorphic gate for analysis is described, and the unconventional features in polymorphic gates that were found are detailed. The results and newly crafted gates is reported in Section 4.3. Section 4.4 discusses and concludes the chapter.

## 4.2 Finding the Sources of Polymorphism

Polymorphic circuits are commonly designed through evolutionary algorithms. To understand where the polymorphism originates, multiple circuits need to be selected for analysis, and commonalities among them need to be identified. A categorization methodology was developed to find representative polymorphic circuits from the circuits that are created by the evolutionary algorithm and those published in the literature. These circuits are analyzed to identify their commonalities, particularly those that do not appear in traditional CMOS gates. This analysis lead to the discovery of a new type of polymorphism.

### 4.2.1 Evolutionary Algorithm

Evolutionary algorithms in this dissertation are algorithms that begin by randomly generating a large pool of candidate circuits, testing the candidate circuits against a fitness function, selecting the candidates that best meet the fitness function, mutating them, and creating the next

generation of candidate circuits. The fitness function used is critical to the correct operation of the algorithm, and is described in detail in Algorithm 1 and figure 4.1.

The fitness function in algorithm 1 is built so that it can detect a circuit that is approaching the target function. For example, if a circuit has output voltages of 3.3V, 0.0V, 0.2V, and 0.3V, that circuit would be classified as a NOR circuit with threshold of 90%. This is because the value furthest from the target (3.3V for the high value and 0 for the low values) is 0.3V, which is 10% of the high value. This algorithm looks at every possible voltage pair for every possible target circuit and calculates the highest possible threshold for the circuit. This threshold is then used to determine if the circuit survives the generation.

Evolutionary algorithms are known to have issues that occur during runs. The two primary issues are related to the fitness function used to select gates for the next generation and the evolutionary algorithm stalling. The stalling issue can be addressed by running the algorithm multiple times with different random seeds, but the fitness function is more difficult to address. The categorization algorithm below is our solution to the fitness function problem.

### 4.2.2 Categorization

The fitness test used in the evolutionary algorithm is based on HSPICE simulation using a fixed library, fixed measurement time, and fixed input order. The algorithm will find circuit results based only on these settings, and it is possible that the resulting circuits are flawed and the polymorphism may disappear when any of the settings change. The following categorization system was created and tests to find representative robust polymorphic gates:

1. Simulator 1 and Library 1: The original test used by the evolutionary algorithm

**Algorithm 1** Evolutionary Algorithm

```
Require: g: number of generations, n: candidates per generation, r: survival rate
for gen in range(g) do
    for i in range(n) do
        x = new CircuitMap(random)
        while not validateMap(x) do
            x = new CircuitMap(random)
        end while
        N.add(x)
    end for
    for x in N do
        Simulate(x)
        calculateFitness(x) {See below for this algorithm}
    end for
    sort(N) {Sort by fitness function}
    for i in range(r*n) do
        x = nextInN {Take the next lower}
        s.add(x) {s now is the set of surviving circuits}
    end for
    childCount = .9*n/s.size {90% of next generation}
    N.clear()
    for i in s do
        for j in range(childCount) do
            x = mutate(i)
            if validateMap(x) then
                N.add(x) {Add only valid children}
            end if
        end for
    end for
    while count(N) ¡ n do
        x = new CircuitMap(random)
        while not validateMap(x) do
            x = new CircuitMap(random)
        end while
        N.add(x)
    end while
end for
FUNCTION calculateFitness(circuit x):
V = readResultingValues(x){Get voltage values for each voltage level}
fitnessValues = [V][T] {initialize the fitness values set}
for t in T do
    for v in V do
        fitnessValues[v][t]=(calcThresh(t,v)) {compute how far the results v are from target.}
    end for
end for
threshold = min(findBestPair(fitnessValues) {Compute the best pair from fitnessValues such that the
voltage and target are not repeated}
if threshold > targetThreshold then
    save(x)
end if
return threshold
```

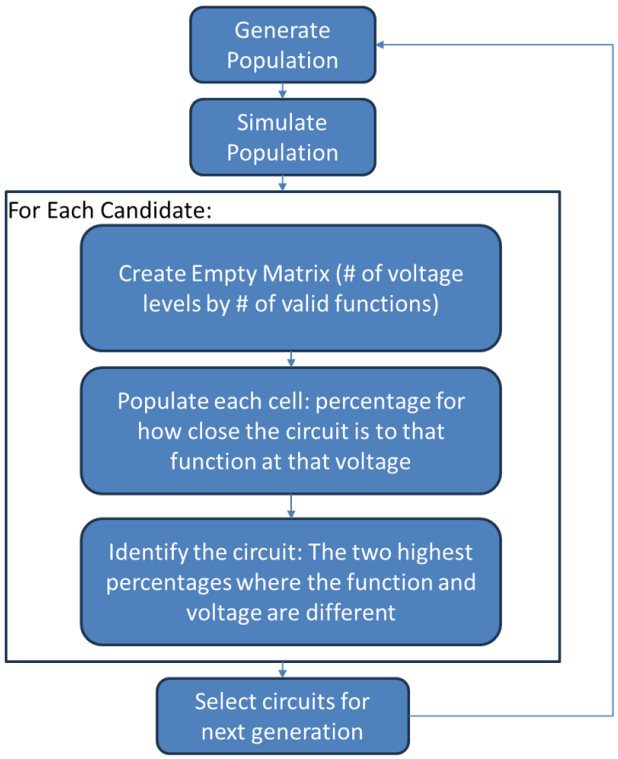



Figure 4.1: This chart details the progression of the evolutionary algorithm inside a generation, with the fitness function being applied to each candidate circuit.

2. Simulator 2 and Library 1: Control for the simulator by using a second simulator from a different vendor

3. Input Invariance: Control for the fixed input sequence, one test with each simulator

4. Input Invariance, Simulator 2: Same test as above with the second simulator

5. Simulator 1 and Library 2: Control for the library by using a different library with the initial simulator

6. Simulator 2 and Library 2: Control for both simulator and library

7. Time Stability: Control for the time given to the circuit to stabilize, i.e., the measurement time. One test with each simulator.

8. Time Stability, Simulator 2: Same test as above with the second simulator

These tests were run on polymorphic circuits generated by an evolutionary algorithm to discover the gates that have the most stable and robust polymorphism. The more tests that a given circuit passes, the higher the likelihood that it possesses robust polymorphism. The first test evaluates the circuit for possible polymorphism using criteria similar to the evolutionary algorithm. The second test is intended to control for a difference in simulator. The third and fourth tests control for a difference in input order. The evolutionary algorithm used for this data tests all four inputs in this order: 00, 01, 10, and 11. This test reverses the order. The fifth and sixth tests control for the library. A failure here by itself is not a failure of the circuit to have polymorphism, but it could indicate that the polymorphism is inherent to the library. If the polymorphism is library dependent, it may be more difficult to determine from a standard circuit analysis. The final two tests (tests 7 and 8) control for the timescale that is used for the polymorphism. The evolutionary algorithm runs on either a millisecond or nanosecond timescale, but this test checks the circuit on the timescale not used by the evolutionary algorithm. For each test after the simulator difference test, a pass with either simulator is deemed a pass for the circuit.

It is believed this method of categorization is reliable in determining if a circuit output from an evolutionary algorithm is polymorphic. However, this methodology is not perfect and does have shortcomings. The first is that not every variable is perfectly controlled: for example, two different simulators are used, but both could perform incorrectly with a particular circuit and the circuit could pass the test. Every possible variable that needs to be controlled for cannot be determined: a series of 8 tests have been proposed, but more may be needed to control for variables that have not been considered. The final shortcoming that has been identified is that the

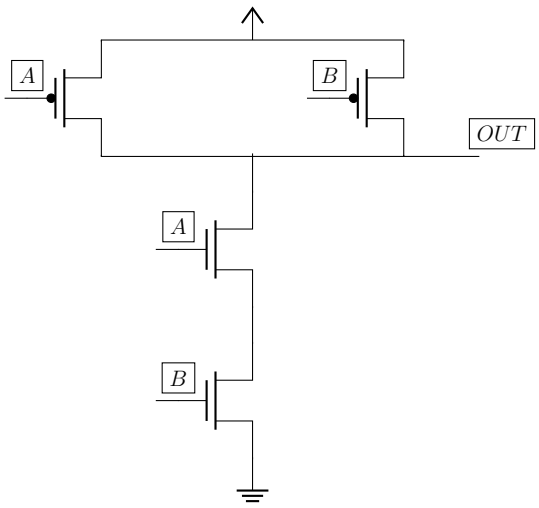



Figure 4.2: Traditional NAND CMOS gate

variables may not be independent: the tests each assume that if a circuit passes both the test with the time-scale changed and the test with the input sequence reversed that it would also pass a test with both the input sequence reversed and the timescale changed. This dependency is not tested and may be present for some circuits.

### 4.2.3 Circuit Analysis Techniques

Before analyzing the circuits that come from the above categorization system, it is important to note the differences between evolved circuits and the conventional CMOS circuits and gates. Figure 4.2 shows the standard CMOS design for a NAND gate. Comparing to the representative polymorphic gates that will be shown next, it can be seen that standard CMOS gates have features that are not present in the following evolved polymorphic gates. CMOS gates are symmetric, do not contain loops, use a standard power source, and have conventional transistor layout and utilization.

## Asymmetry

Standard CMOS circuits are symmetric by design [46]. This symmetry allows for multiple benefits, but one notable for our purpose is that a symmetric design, in general, is easier to understand. An evolutionary algorithm, due to its design, will rarely produce a symmetric output and the circuits should be analyzed with this in mind.

## Loops

A loop is a structure where the output/drain of one transistor feeds and controls the gate of a second transistor, whose output/drain eventually feeds back to the first transistor. An example of this is shown in figure 4.3. Loops can have many possible effects on the circuit, from causing situations where the circuit takes longer to reach a steady state to always pulling the circuit to one value. Loops also can make it difficult to determine precisely what is occurring in the circuit. This difficulty increases with the presence of additional transistors that are part of the loop or connected to it.

## Input-Source

A circuit with an input-source is one where the current that controls the output is sourced from one or more of the circuit inputs instead of $V_{DD}$. Figure 4.4 shows a simple design that illustrates this concept. In traditional CMOS designs, the inputs switch the circuit between outputs, but the main power source, typically $V_{DD}$, drives the output. However, it is noted that evolved circuits may not adhere to this constraint, and they can have inputs that power the output directly. This is an important consideration when incorporating these circuits into larger designs since the

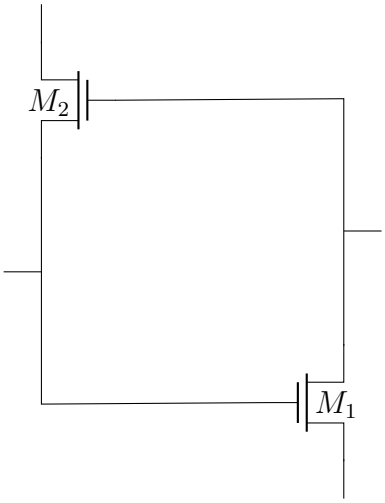



Figure 4.3: This figure shows a simple transistor loop. The loop must be connected to other parts of the circuit at the points indicated.

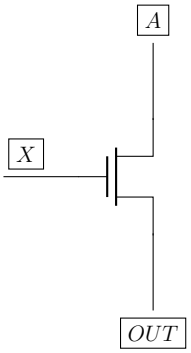



Figure 4.4: Evolved structure showing an input-source. This structure allows an input to directly drive an output in the circuit. Care must be taken to ensure that this input is capable of providing the output with the necessary current.

input signal must be capable of powering the output or the circuit must be placed in front of a buffer [47].

## Unconventional Transistor Utilization

Evolutionary algorithms can use transistors in ways that differ from conventional designs. One example is shown in figure 4.5, where the source and drain of the transistor are tied together. No current flows in this case, but the transistor acts as a small capacitor, combining the capaci-

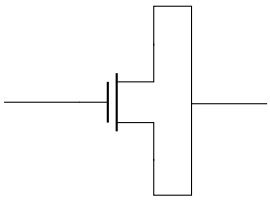


Figure 4.5: Evolved structure showing a transistor acting as a capacitor.

tance $C_{gs}$ and $C_{gd}$. The second unconventional transistor design frequently observed in evolved circuits was noted by [42]: the threshold drop effect. A simple example of this is shown in figure 4.6. This design will force the transistor to act as a diode and reduce the voltage by one $V_{th}$. This design can be chained for more reduction in voltage. Care must be taken when using the threshold drop effect. This effect will force the transistor into subthreshold operation once the source is one threshold away from the gate and drain, but in subthreshold there is still a small amount of current flow. This requires careful consideration by the designer, or mitigation in the evolutionary algorithm. Some mitigation techniques that can be used to address this include ensuring that there is current flowing through the transistor, ensuring that the input to the threshold drop effect turns off part of the time (for example, if an input is connected to the threshold drop transistor), or by ensuring that the leakage current can address the subthreshold current flow. Another effect of this is that switching the circuit from a state where the threshold drop transistor is high to low can cause the circuit to not switch function, since the excess charge cannot be removed.

## 4.3 Results and Analysis

The evolutionary algorithm was run, generating over 13,000 candidate circuits. These circuits were categorized, and a candidate circuit was selected. That candidate circuit was analyzed in an attempt to understand how evolutionary algorithms can discover polymorphic circuits, and

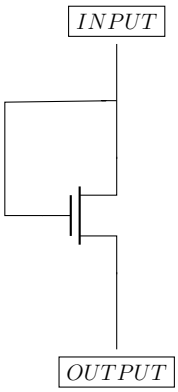



Figure 4.6: Unconventional transistor design showing threshold drop effect. This effect causes the output to be approximately one threshold value lower than the input.

what transistor structures can yield polymorphism. The techniques learned from this were applied to an existing, published circuit, and the results are given below.

### 4.3.1 Categorization

The 13,000 candidate circuits generated by the evolutionary algorithm were subjected to the categorization algorithm using Spectre as the second simulator. The circuits were tested with two input sequences: 00, 01, 10, 11, and its reverse. The second library used for this test was a modified version of the initial library to include more idealized transistors. The timescales used were 5ms and 5ns, with initial runs using 5ns in the evolutionary algorithm and later runs using 5ms in the evolutionary run. The results of the categorization process are presented in Table 4.1.

From the ”Library Independent” category, a circuit was selected for further analysis, shown in Figure 4.7. This evolved circuit was selected because it is less dependent on the library and simulator used, making it less likely to have errors due to the fitness function of the evolutionary algorithm.

Table 4.1: Evolved Circuit Categorization Results

| Category | Criteria | Circuit Count |
|---|---|---|
| None/Simulator Error | No Tests Passed/Simulator Crashed | 4.88% |
| Polymorphism Detected | Polymorphism Simulated | 56.88% |
| Simulator Independence | Polymorphism with multiple simulators | 29.16% |
| Input Independence | Polymorphism with input order changed | 9.00% |
| Library Independent | Polymorphism with multiple libraries | 0.07% |
| Time Independence | Polymorphism with altered timescale | 0.00% |

### 4.3.2 Circuit Analysis

An analysis was done on the circuit in figure 4.7 from our evolutionary algorithm, as well as the circuit in figure 4.8 published by NASA's JPL [1]. The analysis goes through the polymorphism expected and any structures that are present that were mentioned in section 4.2.3. Further analysis of other published circuits is summarized in table 4.2. From this table it can be seen that many polymorphic circuits share the identified structures, which helps confirm the hypothesis that these structures contribute to polymorphism.

#### Evolved Circuit

The evolved circuit exhibits **OR** functionality in the standard state and **AND** functionality in the polymorphic state.

##### Structures

Many structures presented earlier are present in this circuit. Loops are visible in M1 and M2 as F1, in the simplest case, and also appear elsewhere in the design in more complex cases, such as M6, M3, and M2. There are also two transistors, M4 and M5, that are in an unusual

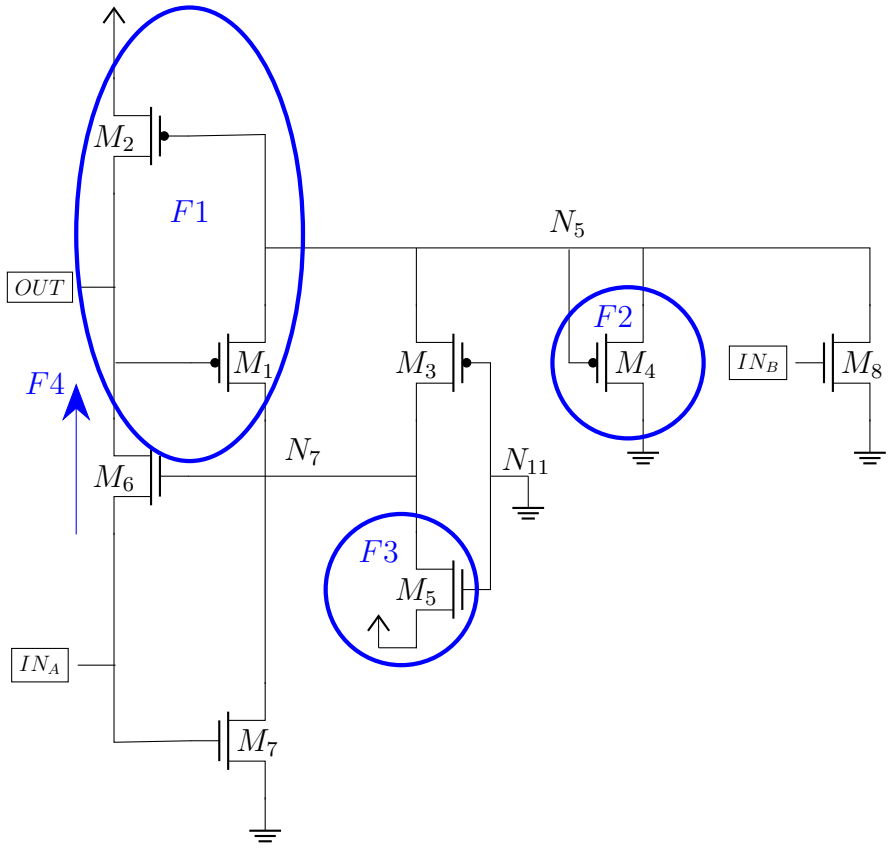



Figure 4.7: The schematic of the evolved circuit that fits into the ”independent” category. This circuit has all four of the discovered structures present. A loop in F1, F2 and F3 show unconventional transistor design, and F4 shows an input driven output.

configuration. M4 acts as a small capacitor on N5, F2, and M5 is always turned off because N11 is grounded in F3. An input-source can also be seen as F4 from input A through M6 to the output.

Analysis

The primary driving transistor for this circuit is M2, which is controlled by N5. If input B is high, N5 will eventually be pulled to a low voltage, changing the output to high voltage. The speed at which N5 is driven low depends on the width and length values of M8. If input A is high, M7 is turned on, pulling N7 low. This allows N5 to be pulled low by both M1 and M3 at the beginning, although it is slowed down by M4 acting as a capacitor. After enough time has

Table 4.2: Overview of Structures Present in Polymorphic Circuits

| Paper | Figure | Signal | Functions | Loops | Unusual Transistors | Input-Source | Asymmetry |
|---|---|---|---|---|---|---|---|
| [49] | 6 | $V_{DD}$ | OR/AND | X | X | X | X |
| [1] | 1 | Temp | AND/OR | | X | X | X |
| [1] | 3 | $V_{target}$ | OR/XOR/AND | | X | X | X |
| [1] | 4 | $V_{DD}$ | AND/OR | X | | X | X |
| [42] | 3 | $V_{DD}$ | Majority/AND | | X | X | X |
| [4] | 2 | Temp | AND/!b | | X | | X |
| [2] | 3 | $V_{target}$ | AND/OR | | | X | X |

passed, the output will be high enough to turn off M1, which will further slow down N5 being pulled low.

## Published Circuit

The circuit published by JPL in [1](their figure 3) and is redrawn in figure 4.8 to aid in the identification and analysis of structures of interest. The circuit is stated to exhibit functionality based on the voltage of $V_{target}$ and referenced in table 4.3. It is important to note that for this circuit, polymorphism is based on target voltage, not $V_{DD}$, and $V_{DD}$ is held at a constant 3.3V.

### Structures

The circuit has two primary structures present that are of note. The first is that this circuit is input-powered, shown in F1. This means that there is no clear way for a $V_{DD}$ connection to move to the output. The second item of note is that there is a threshold-drop transistor present at M10 in F2, and M8 acts as a capacitor in F3. The threshold-drop transistor shown in F2 is controlled

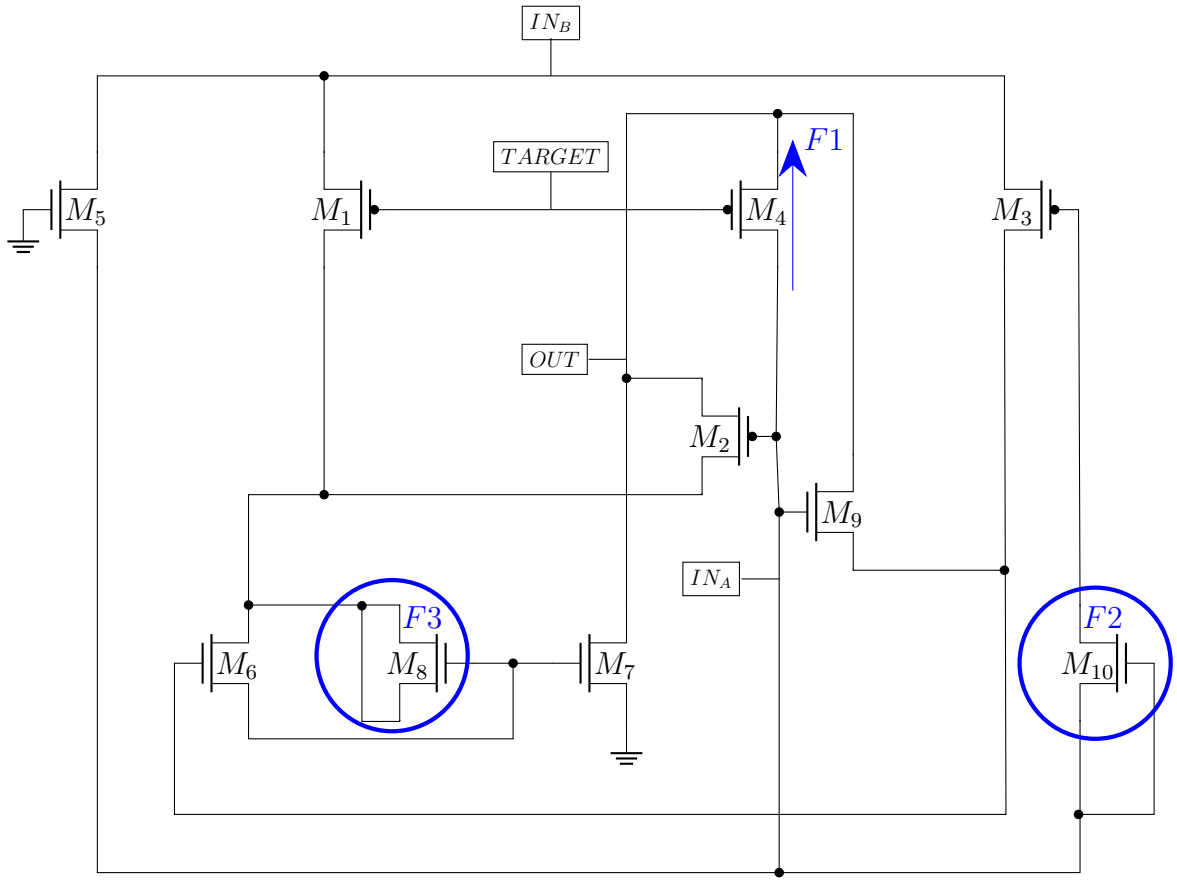



Figure 4.8: This shows the schematic of the redrawn circuit from JPL [1]. The JPL circuit has many of the discovered features present: F1 shows the input driven output, F2 and F3 both show unconventional transistor design.

by input A, which means that the excess charge on the gate of $M_3$ can be removed when input A is low.

Analysis, $V_{target}$ = 0V

The first case to analyze is where $V_{target}$ is 0V. The circuit acts as an OR circuit. When A is high, M4 makes a clear path to the output. When input B is high and $V_{target}$ is low, if A is also low then M1 and M2 make a path from input B to the output. When A is also high the previous situation occurs allowing a path from A to out.

Table 4.3: Target Circuit Functionality as Stated by JPL [1]

| $V_{target}$ | Functionality |
|---|---|
| 0V | OR |
| 1.5V | XOR |
| 3.3V | AND |

Analysis, $V_{target}$ = 1.5V

Next the XOR case is looked at. The inputs of interest here are when both A and B are high. When this occurs, M4 would still be pushing A to the output, but it is hindered by the gate voltage on M4. At the same time, M9 is pulling current from the output. If the output is high enough, this will turn on M6. M1 is on because of the target voltage, and this allows M6 to turn on M7, pulling the output to ground. This causes a balancing act whose timing is determined by the parameters of the transistors, which are not published.

Analysis, $V_{target}$ = 3.3V

In the last case, the circuit is an AND gate. In this case, M1 and M4 are turned off completely. This causes the only path for any input to reach the output to be from input B, through M3 and M9. M9 is turned on by input A, and M3 is turned on by A through M10, which provides a threshold drop effect. This causes the output to be high if and only if both A and B are high, thus resulting in an AND gate's functionality.

### 4.3.3 Created Polymorphic Gates

The structures that have been discovered can be used to create new polymorphic gates. The first example gate presented is a polymorphic NAND/NOR gate, shown in figure 4.9. This gate

Table 4.4: Target Circuit Functionality for Custom NAND/NOR gate

| A | B | Out 1.2V | Out 3.3V |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

Table 4.5: Target Circuit Functionality for Custom B buffer/OR gate

| A | B | Out 1.2V | Out 3.3V |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |

uses the threshold-drop effect applied twice to $V_{DD}$. This pulls the signal "SEL" low when $V_{DD}$ is 1.2V and high when $V_{DD}$ is 3.3V. The design of this circuit must be done in such a way that the subthreshold current does not allow the voltage on the SEL line to increase above $V_{TH}$ unless the circuit is operating in that mode. In this particular case, transistor design parameters can be modified to allow for correct operation. The steps taken to design this circuit are as follows:

1. Design the truth table for the target gate, shown in table 4.4.

2. Identify common outputs across all target voltages, in this case the first and last input

3. Design the structures representing the common functions: M6, M7, M9, and M10 in figure 4.9.

4. Build the selection signal, in this case using the threshold drop effect on $V_{DD}$, shown with M1 and M2.

5. Design the structures that show different outputs, placing them behind the masking transistors controlled by the select signal.

The second created polymorphic gate is shown in figure 4.10. This gate represents an OR gate and a B buffer, as shown in table 4.5. This gate utilizes both an input-source as well as two threshold drop transistors. This circuit has the interesting property of masking the A input when the voltage is low, and being an OR gate when voltage is high.

The first created polymorphic gate uses a system that allows creation of a wide array of functions, but it is important to note that as the number of common outputs decreases, more transistors must be masked by the select signal. In the extreme case, this approaches a polymorphic multiplexing strategy: a signal simply switches between two different gates. The second created gate can be used to mask any signal or set of signals from a given gate, the A input in this case. The design of the above gates shows that some of these structures can be used to craft polymorphic gates without the use of evolutionary algorithms.

## 4.4 Discussion

This chapter presented some of the structures that evolutionary algorithms generate when searching for polymorphic circuits. Notable structures include loops, input-powered outputs, and unusual transistor placements. Evolved circuits are rarely symmetric, and understanding these structures can help designers modify or create polymorphic circuits more efficiently, without the need for running an evolutionary algorithm each time.

A categorization method for evolved circuits was also presented, which can help designers determine if the circuit was subject to fitness function error. This categorization method informed

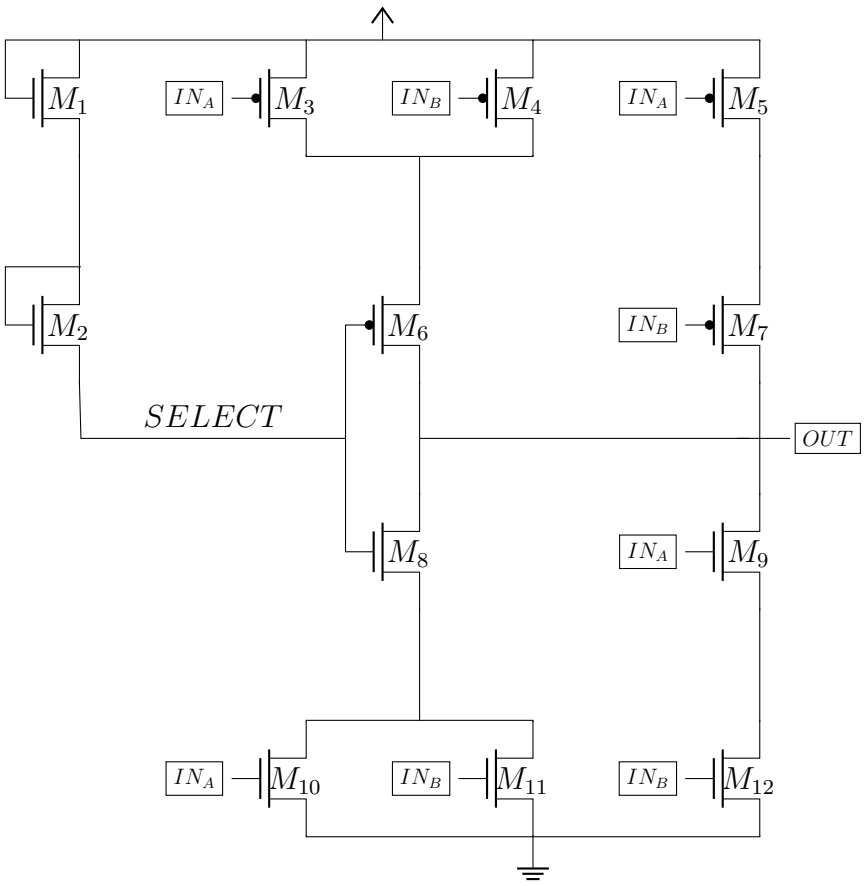



Figure 4.9: This shows the schematic of the custom designed NAND/NOR polymorphic circuit. This circuit is built using two threshold drop transistors (M1 and M2 in the top left) connected to $V_{DD}$. This selectively enables transistors M3 and M4 or M10 and M11.

the analysis methodology for the evolved polymorphic circuits analyzed in this paper, including the circuit previously published by NASA's Jet Propulsion Lab, as well as a circuit discovered by the authors.

Using the structures identified in the earlier sections, two polymorphic gates were manually designed. The first gate is a polymorphic NAND/NOR gate, created by applying the threshold drop effect with two stages to $V_{DD}$ and using the result to mask or unmask pieces of the logic. This allows the reuse of transistors that trigger the common outputs, while splitting off the transistors that trigger the polymorphic outputs. The second created gate is used to mask a particular input. The gate presented is a B passthrough when the A input is masked, or an OR gate when both

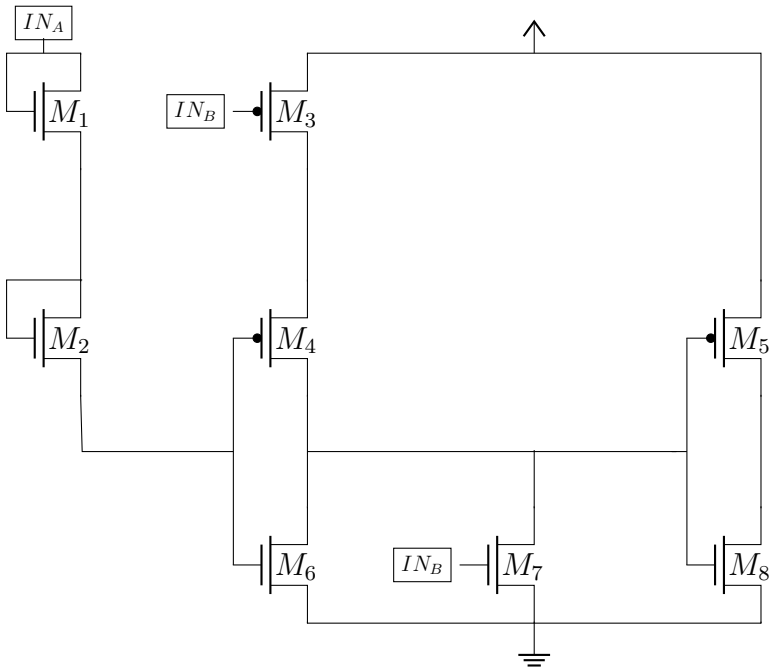



Figure 4.10: This shows the schematic of the custom designed OR/B polymorphic circuit. This circuit is built using two threshold drop transistors (M1 and M2 in the top left) connected to the A input.

inputs are considered.

# Chapter 5: Time-Based and Multi Polymorphism

## 5.1 Chapter Overview

Polymorphic circuits are circuits that exhibit different behavior under different conditions. The different conditions used to select the circuit function are most commonly voltage, temperature, or an external signal [1], but some circuits are dependent on another factor as well: time. Time-dependent circuits will exhibit polymorphism based on an outside factor, but only when the sample rate of the circuit is correct. This is time-based polymorphism. It is similar to multi polymorphism, where the same signal is used to switch between more than two different functions.

Specifically,the following contributions are made in this chapter:

- A new type of polymorphic circuit that is controlled by clock frequency or time is introduced, which is referred to as time-based polymorphism.
- Multi polymorphism is presented, where a circuit exhibits more than two visible functions.
- Multiple potential applications of time-based and multi polymorphism are given.

The rest of the chapter is organized as follows: section 5.2 presents a detailed analysis of time-based and multi polymorphism, both new circuits and ones published in existing literature. Section 5.3 gives the results of the evolution algorithm, categorization data, and details of the

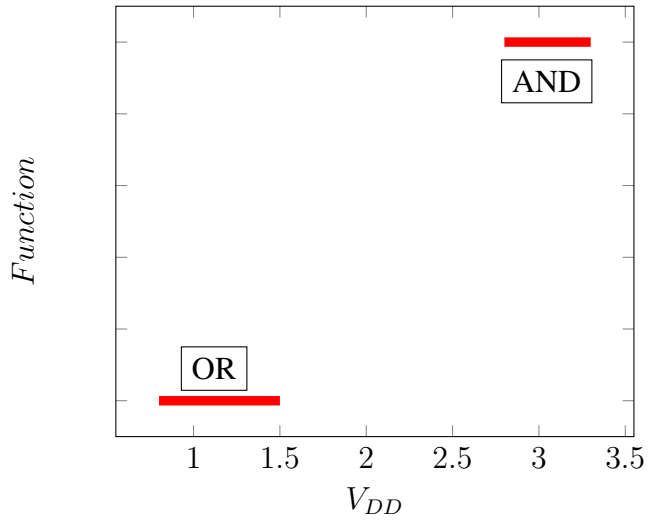



Figure 5.1: This graph gives a 2-dimensional example of a polymorphic function graph. In this example, the function is determined by $V_{DD}$.

functions in the presented circuits. Section 5.4 presents hardware-security applications for time-based polymorphism, and section 5.5 concludes the paper.

## 5.2 Principals of Time-based and Multi Polymorphism

In this section both time-based polymorphism and multi polymorphism are presented in the generic form. Each subsection includes a description as well as any references in existing literature. A method to better describe polymorphism based on a multi-dimensional space is also presented, where the independent variables are the polymorphic control signals and the resulting curve is the function. A simple two dimensional graph, where only $V_{DD}$ controls the function, is given in figure 5.1. In this example, when $V_{DD}$ is at 1.2V the function is OR, at 2.1 the function is XOR, and at 3.3V the function is AND. There is also no guarantee that the function will be continuous or clearly defined. The function graphs are intended to be a tool to represent graphically the functions and the variables that control them.

### 5.2.1 Time-based polymorphism

Time-based polymorphism is polymorphism that relies on how quickly the circuit output is sampled from the inputs changing. Possible candidates for time-based polymorphism are circuits that fail the categorization test for timescale, but pass all other relevant tests, as detailed later. These circuits require a set, known state to return to after the circuit is sampled, so they are presented as synchronous circuits that should be reset on the off cycle of a clock signal.

The time-based polymorphic circuits can be combined with another polymorphic type, such as $V_{DD}$. Table 5.1 shows a comparison between a standard polymorphic circuit and the new time-based polymorphic circuit described. In this form of time-based polymorphism, standard circuit functionality is shown as ”function 1” and is present in 3 of the 4 cases, only showing the alternative function, ”function 2”, when the circuit voltage is dropped low, and the circuit is sampled quickly. This leads to some interesting potential applications that will be discussed later in the applications section of this chapter, section 5.4.

A graph showing this type of polymorphism is given in figure 5.2. In this graph, the independent variables are time and voltage, with function 1 being the low result and function 2 being the high result. The area in the middle, though continuous in this example, is not necessarily well defined and can be omitted if desired, unless intermediate steps are known.

The core principals of time-based polymorphism, as presented, are given below:

- Time-based polymorphism augments a different type of polymorphism, such as polymorphism based on $V_{DD}$ and therefore is at least three dimensional on the polymorphic function graph.

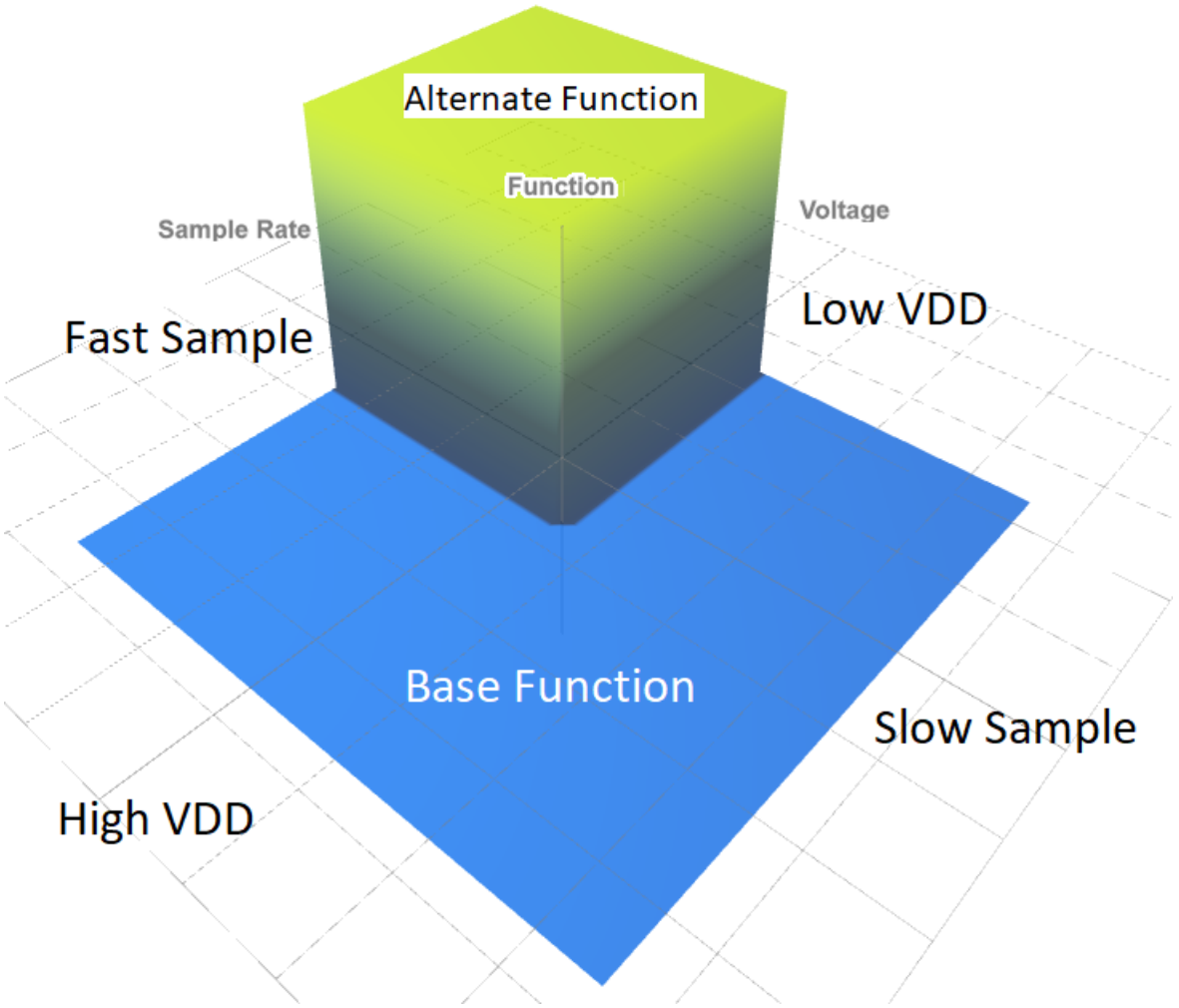



Figure 5.2: This graph gives a 3-dimensional example of a time-based polymorphic function graph. The function is determined by sample rate and $V_{DD}$.

- Time-based polymorphism depends on the sample rate of the circuit.
- Time-based polymorphism does not have to reveal a new polymorphic function, as it can just mask an existing function.

Table 5.1: Time-Based Polymorphic Comparison

| State | Traditional | Time-Based |
|---|---|---|
| Slow Sample, High $V_{DD}$ | Function 1 | Function 1 |
| Slow Sample, Low $V_{DD}$ | Function 2 | Function 1 |
| Fast Sample, High $V_{DD}$ | Function 1 | Function 1 |
| Fast Sample, Low $V_{DD}$ | Function 2 | Function 2 |

### 5.2.2 Multi Polymorphism

Multi polymorphism is where the same signal, such as $V_{DD}$, has more than two valid levels. Polymorphism such as this is given in [1], recreated in figure 5.3, where the functionality of the circuit switches between OR, XOR, and AND depending on the voltage of a trigger signal.

Multi polymorphism, in its most basic case, can be seen in the two dimensional polymorphic function graph in figure 5.4. This graph shows the three different levels presented in [1] for the circuit redrawn in figure 5.3.

The core principles of multi polymorphism are as follows:

- Polymorphic circuit has at least three functionality levels
- The three levels do not need to be unique, as long as they differ from immediate neighbors

## 5.3 Results

### 5.3.1 Categorization

The objective of categorization is different for time-based polymorphism. In traditional polymorphism, the objective is to have as many tests pass as possible to help ensure as robust

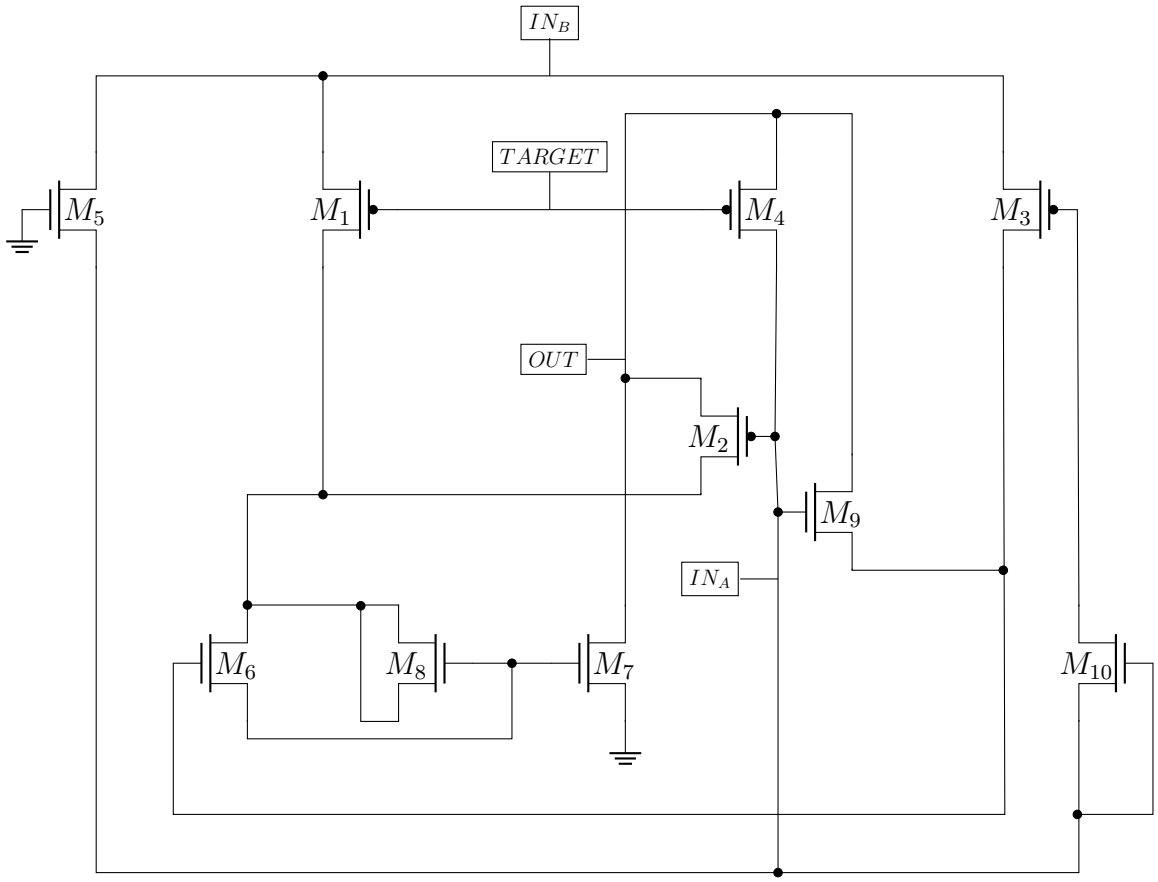



Figure 5.3: This shows the schematic of the redrawn circuit from JPL [1]. This circuit is a published example of a circuit with multi polymorphism, with three functions: OR, XOR, and AND.

a polymorphic gate as possible. In time-based polymorphism, the objective is for the gate to be simulator and input-order independent, but to change with the time-scale.

### 5.3.2 Multi Polymorphism

Multi polymorphism is exemplified by circuits that are designed using the principles given in [49]. In figure 5.5, which is a modification of the custom circuit presented in [49], there are multiple situations where a transistor is used to drop the voltage. The circuit here has been modified by adding another threshold drop transistor, M3, and splitting out the control transistors into one per input. This allows us to generate a new function: NOT B.

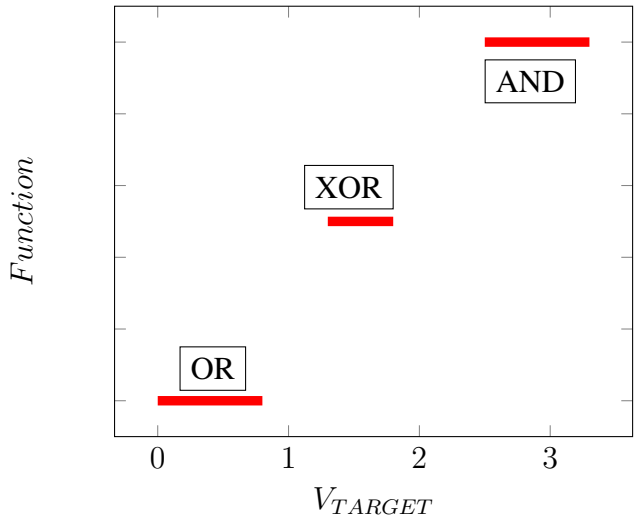



Figure 5.4: This shows the function map of the circuit from JPL [1]. This circuit has functionality based on $V_{TARGET}$ and switches between OR, XOR, and AND.

Table 5.2: Custom Multi Circuit Functionality

| $V_{DD}$ | **Functionality** |
|---|---|
| 1.2V | NAND |
| 2.5V | NOT B |
| 3.3V | NOR |

The new functionality of the circuit is given in table 5.2. The function map for this circuit is given in figure 5.6.

### 5.3.3 Time-Based Polymorphic Circuits

The evolved circuit in figure 5.7 from [49] demonstrates time-based polymorphism. The results can be seen in figure 5.9, which shows timing for when only A is high, and in figure 5.10, which shows timing when only B is high. This circuit exemplifies time-based polymorphism both before figure 5.9 goes high at around 10ns and after this point but before figure 5.10 goes high. The exact functionality is shown in Table 5.3. Ultimately, this function also presents multi polymorphism, since it operates under three functions depending on the time-scale used to sample

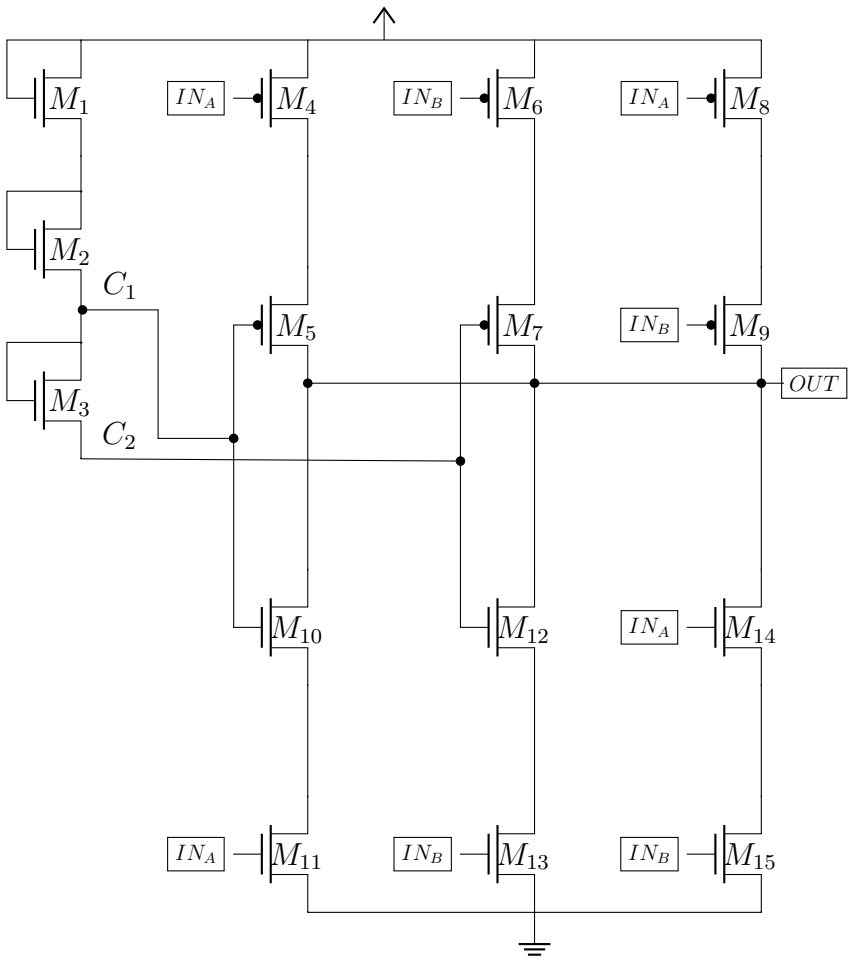



Figure 5.5: This shows the schematic of a custom multi polymorphic circuit. This circuit is a modified version of one presented in [49] with functions NAND, NOT B, and NOR.

it. The polymorphic function graph for this circuit is shown in figure 5.8

## 5.4 Applications

Applications for polymorphic circuits have been discussed and realized in other papers [15, 50]. Some of these applications are relevant to hardware security [39], and time-based polymorphism is well suited for these applications. Three possible applications are introduced in this section, embedded signatures, hidden functionality, and anti-reverse engineering.

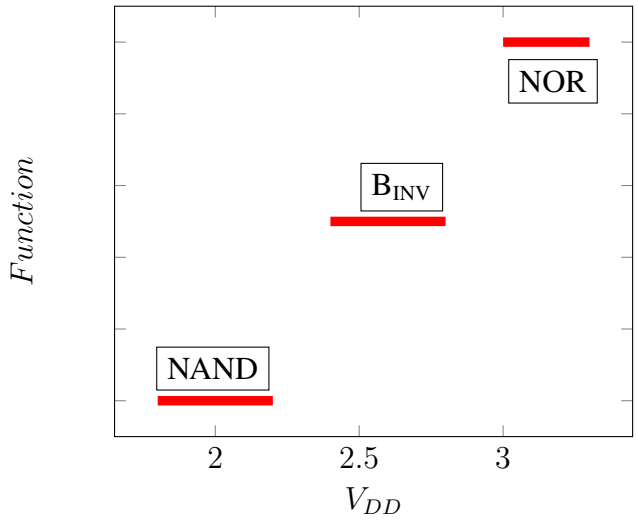



Figure 5.6: This shows the function map of the custom multi polymorphic circuit. This circuit has functionality based on $V_{DD}$ and switches between NAND, NOT B, and NOR.

Table 5.3: Evolved Circuit Functionality at $V_{DD}$=1.2V

| **Sample Rate** | **Functionality** |
|---|---|
| Under 5KhZ | OR |
| 500KhZ-50MhZ | B |
| Over 500MhZ | AND |

### 5.4.1 Embedded Signing and Hidden Functionality

An embedded signature or watermark is a mark or pattern built into a design to allow the creator to prove that they created the design [51]. The standard polymorphic gate has been used previously to place a signature/watermark on circuits [4], and this one would be used in the same way. The additional contribution of this method is that it makes it more difficult for an adversary to discover the watermark since both the timing and the voltage would need to be identified.

Time-based polymorphism also provides the ability to embed additional functionality into a design. The presented circuit has three functionalities: OR, B, and AND. These functionalities can be used to replace other gates to create an internal hidden function.

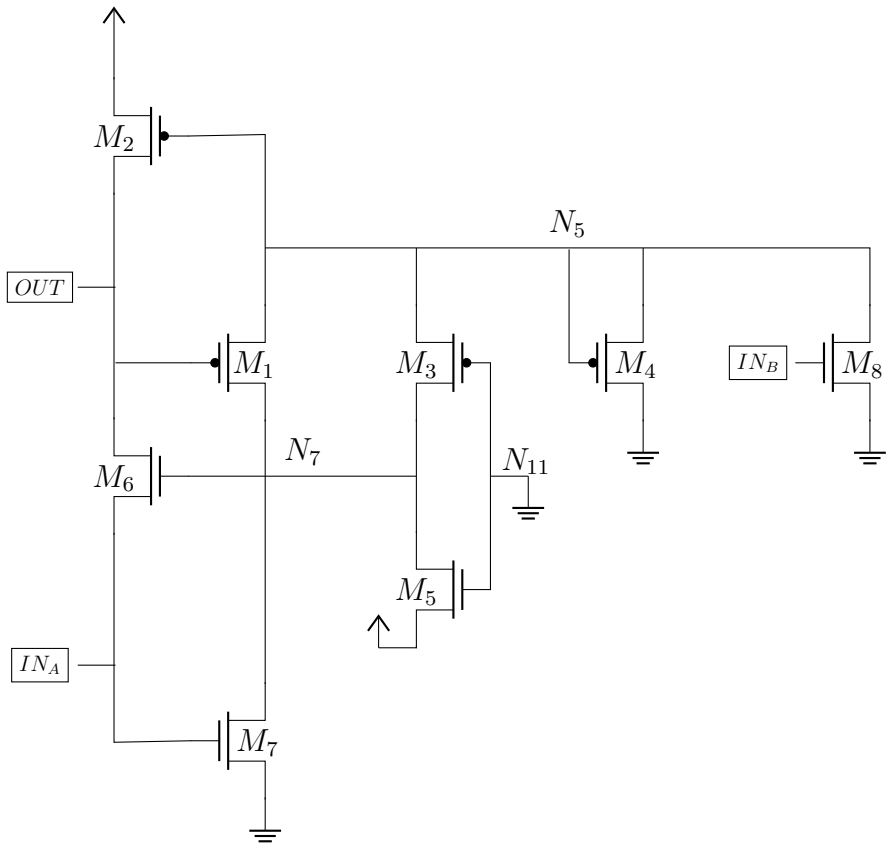



Figure 5.7: This shows the schematic of the evolved circuit. The circuit exhibits time-based polymorphism when the sample rate changes from 5kHz to 500kHz and to 5MHz.

### 5.4.2 Reverse Engineering Prevention

Reverse engineering is one of the most significant challenges to IC fabrication [52]. Time-based polymorphism can also be used as a technique to prevent some hardware reverse engineering attacks. Reverse engineering attacks are frequently carried out with access to the netlist or photographs taken of each device layer [53]. Since these polymorphic gates are evolved, they do not exist in the standard libraries used to carry out this reverse engineering. This means that some of the analysis would either need to be carried out by running the chip, or through simulation. In either case, the speed that the attacker uses will change the result given with time-based polymorphism. This could lead the attacker to have difficulty in determining the exact function

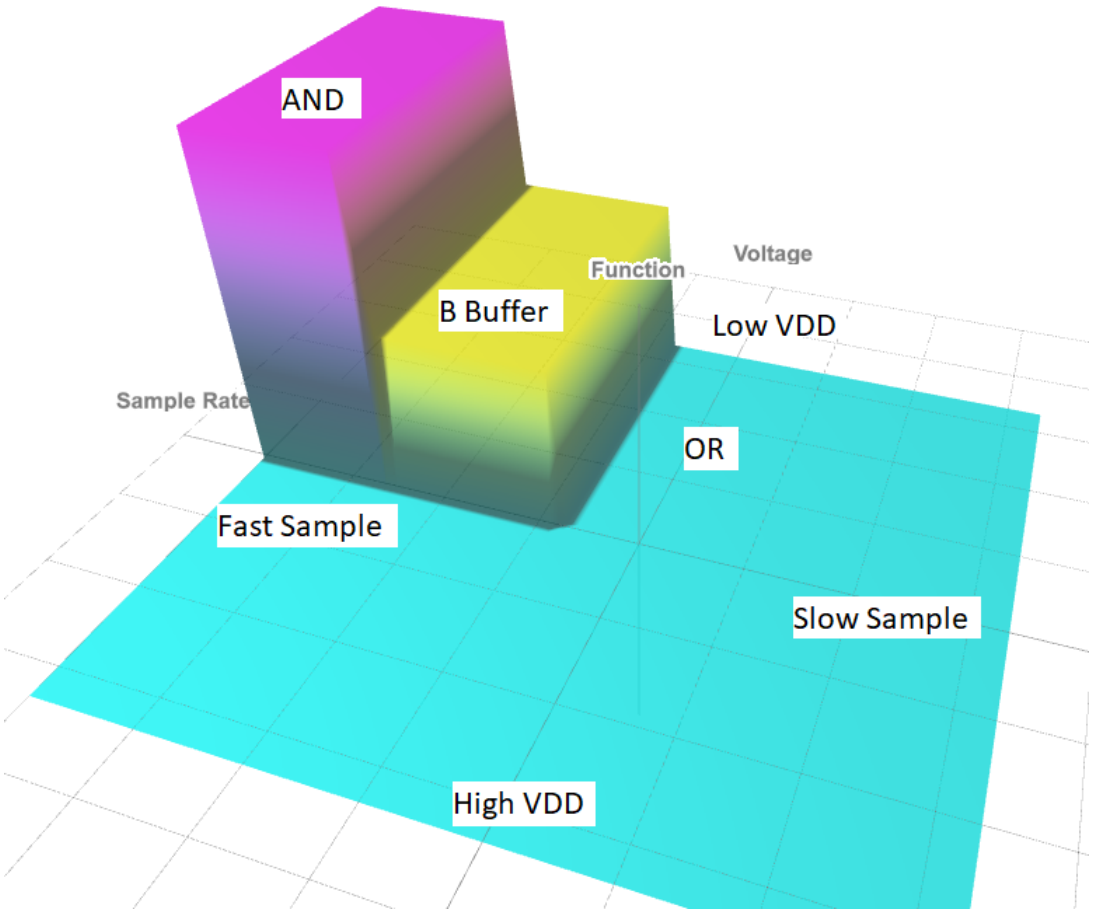



Figure 5.8: This shows polymorphic function graph for the evolved time-based polymorphic circuit in [49]. This also shows multi polymorphism, since it operates with more than 2 functions.

of the device under test.

### 5.4.3 Compliance

Multi-polymorphism can also be applied to enforce compliance with operating specifications. In a multi-polymorphic circuit with a single control variable, the polymorphic function can be visualized as a one-dimensional curve representing the circuit's behavior under different control values. This curve can have multiple stable states corresponding to distinct circuit functionalities.

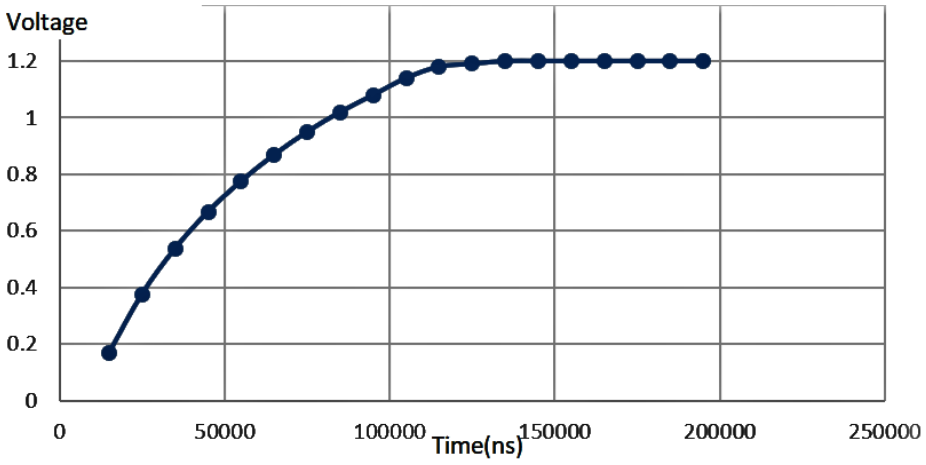



Figure 5.9: This shows the output voltage of the evolved circuit shown in figure 5.7 when $V_{DD}$ is 1.2V and A is high. Note that the timescale is in ns.

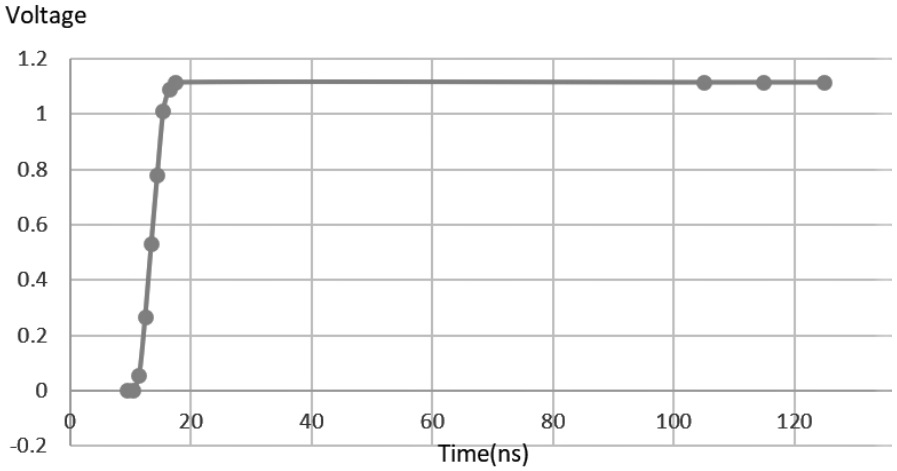



Figure 5.10: This shows the output voltage of the evolved circuit shown in figure 5.7 when $V_{DD}$ is 1.2V and B is high. Note that the timescale is in ns.

By strategically designing the circuit, a specific ”compliant” function can be positioned as the central state within the function space. The circuit can be engineered such that deviations from the chosen control value (moving away from the central state on the function curve) lead to a significant degradation or complete loss of functionality in the neighboring states. This essentially forces the circuit to operate in the compliant state for the system to function correctly.

## 5.5 Discussion

In summary, this chapter has introduced and explored the concepts of time-based and multi-polymorphism in circuits. Time-based polymorphism refers to the phenomenon where a circuit's functionality depends on the speed at which its output is measured. Multi-polymorphism, on the other hand, allows a single circuit to exhibit more than two distinct functionalities based on either a single or multiple control signals. To effectively visualize these concepts, a novel graphing system was presented for representing the polymorphic function space.

These forms of polymorphism have opened doors to several potential applications, particularly for time-based polymorphism. These applications include embedding hidden signatures for security purposes, concealing additional functionalities within circuits, and hindering reverse engineering efforts. By incorporating signatures within the slow, low voltage functionality of a time-based polymorphic design, detection becomes significantly more challenging. Additionally, the ability to strategically place functionalities at different speed levels allows for the creation of hidden features within the circuit. Furthermore, the evolved and non-standard nature of time-based polymorphic circuits makes it difficult for attackers to determine the circuit's exact function during reverse engineering attempts.

Multi-polymorphism also offers its own set of applications. One such application, briefly discussed in this chapter, is compliance enforcement. In this scenario, a circuit can be designed to operate only within a specific range of accepted values, ensuring the correct manifestation of the desired polymorphic function. For instance, a multi-polymorphic circuit controlling a power supply could be designed to limit the output voltage to a safe range, preventing potential damage to other components.

By incorporating these suggestions, you can elevate the formality and impact of your conclusion while providing a clear and concise summary of your research on polymorphism in circuits.

# Chapter 6: Enhancing Subthreshold Stuck-at Fault Testing with Polymorphic Gates

## 6.1 Chapter Overview

With the continuing technology scaling, the increasing of logic density on the chip, and the growing complexity in design and fabrication, the ability to test a circuit's manufacturing defects after fabrication is critical. This is even more important in the subthreshold domain where faults are more prevalent [30]. The stuck-at fault model simulates manufacturing defects by assuming that individual signals to logic gates remain at a constant value due to these defects. It has been well-researched and widely adopted in commercial design and test tools such as automatic test pattern generation.

Take a 2-input AND gate $z = AND(x, y)$ for example, if the input line $x$ has a defect and the signal remains at logic high (1), which is called stuck-at-1, one can detect it by the input pattern $(x = 0, y = 1)$. This is because the test pattern should generate a logic 0 at output $z$, but the stuck-at-1 fault forces the output to be 1. Given the size and depth of the modern circuits, generating the test patterns needed to identify faults is difficult, and impossible in certain cases. Indeed, finding the minimal number of test patterns to provide full coverage of all the faults in a given circuit is known to be NP hard [48].

Some faults are more difficult to discover than others. To address this, the concept of replacing some of the standard gates in the circuit with polymorphic gates is presented. This work is motivated by the need to find better methods to test for faults, particularly in the subthreshold region, but this method applies to both above and subthreshold operation. There are two primary contributions in this work:

1. Enhancing testing in subthreshold with the replacement of a subset of gates with polymorphic gates is presented

2. Generating polymorphic gates in subthreshold with a faster convergence vs traditional evolutionary algorithm design

The paper first provides background on polymorphism, stuck-at fault testing, and the challenges of circuit testing. It then explores how polymorphism can improve fault detection, showcasing the concept with both complete gate replacement in a D Flip Flop and partial replacement in a C17 benchmark circuit. These examples demonstrate the generalizability of the approach to various circuit designs. Finally, the paper discusses modifications to a standard evolutionary algorithm to accelerate the search for subthreshold polymorphic gates, leveraging known polymorphic structures [49].

## 6.2 Evolutionary Algorithm

The evolutionary algorithm detailed in Algorithm 1 in section 4.2.1 was modified for subthreshold polymorphic circuit discovery. Two types of polymorphic gates were sought with the evolutionary algorithm with the first being high voltage to subthreshold, which used 0.5V and 3.3V as $V_{DD}$. Low voltage to subthreshold was also investigated using three $V_{DD}$ of 0.4V, 0.7V,

and 1.0V. Because of the flexibility of the original algorithm, this only required changes to the configuration of the algorithm, switching the target voltages from above-threshold to subthreshold levels. For this test, a valid result is defined as results that passed the initial two tests in categorization: testing for simulator differences, as well as the test that varies the input order.

Another modification that was made for the low-voltage subthreshold algorithm was seeding of structures. Through research conducted and reported in [49] and section 4.2.3, the input-source structure was selected to seed into the algorithm. This structure exists when an input, A or B, has a path to the output going through transistor source/drains only. In order to seed the input-source structure, each circuit that is randomly generated by the algorithm is required to have an input-source or else it is discarded without simulation. It is, however, possible for the structure to be evolved away. The algorithm was run 90 times, 45 times with seeded mode and 45 with standard mode and with all other parameters kept identical, for low-voltage subthreshold gate creation. The result was that a total of 8 runs in seeded mode and 4 runs in standard mode produced output gates that passed categorization at least through the other simulator and switching the input order. Though the sample size is fairly small, the result is that a seeded algorithm performed notably better in searching for valid gates, and this will be explored further in section 6.4.

After these modifications were made, the evolutionary algorithm was run searching for high-voltage subthreshold polymorphism, as well as two runs of low-voltage polymorphism, one input-source seeded and one not seeded. Initially, the evolutionary algorithm was expected to find valid gates with higher frequency in subthreshold. This is because transistors in subthreshold have an exponential I-V curve in subthreshold, shown in figure 6.1, as opposed to above-threshold operation. The actual results are not as expected, indicating an independence from the I-V curve

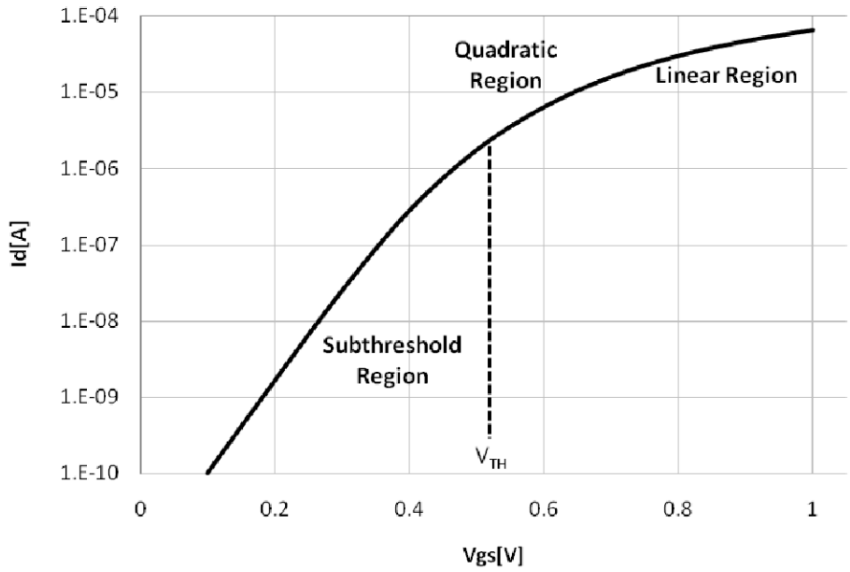



Figure 6.1: This figure shows the IV curve of a MOSFET transistor in both above and subthreshold operation [28].

and resulting in the unexpectedly low number of valid results.

## 6.3 High Voltage and Subthreshold Voltage Polymorphism

The first form of subthreshold polymorphism that was searched for was high-voltage subthreshold polymorphism with $V_{DD}$ between 0.5V and 3.3V. These voltages were chosen because they represent a subthreshold voltage that is still relatively high, 0.5V, and the upper limit to what the evolutionary algorithm has searched for to this point, 3.3V.

All of the gates found by the evolutionary algorithm were put through categorization, where they were evaluated for quality and for any issues with the evolutionary algorithm, and the gates that met the highest categorization for each run were analyzed for power consumption. Each of these tests are reported below in detail. An example circuit is shown in figure 6.2.

Applications for this type of subthreshold polymorphism are limited because of the low power nature of subthreshold devices [54]. These devices would require a 3.3V supply to switch

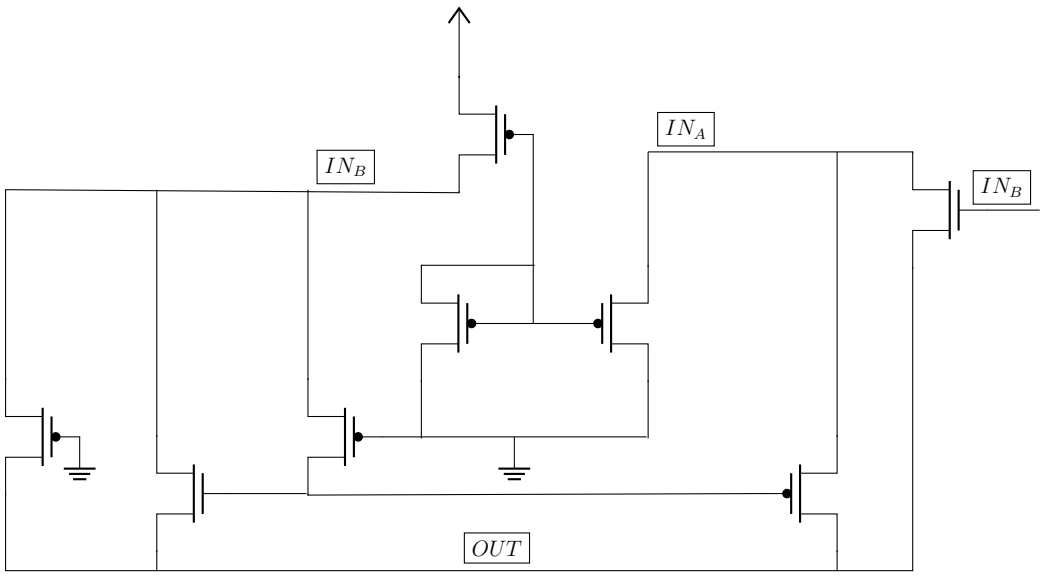



Figure 6.2: This figure is an example high-voltage subthreshold polymorphic gate discovered by the evolutionary algorithm.

the function, but some possible applications to exist. It could be used for a manufacturing test procedure, where the actual device would not need to operate at the higher voltage, but the chip can be tested for faults using the higher voltage. It could also exist to hide a watermark, since during nominal operation this would not need a high voltage supply, and high voltage would only be necessary for revealing the security feature.

### 6.3.1 Categorization

The evolutionary algorithm was run 57 times. The total number of candidate gates across all 57 runs was 20,787 gates, though many of these gates are functionally the same with very minor changes, such as a slightly different W or L value. Because of this, it is important to instead look at the statistics present in a given run. Of these 57 runs, 15 runs produced candidate gates. These candidate gates were simulated with both Synopsys HSPICE and Cadence SPECTRE to check for simulator dependent errors, input order dependence, library dependence, and time dependence.

This categorization run had 74% of runs fail to converge on a suspected polymorphic circuit, with 80% of the remaining runs, making up 21% of the total runs, converging on at least one circuit that was input independent. The candidate circuits that passed the ”Input Independence” test and ”Simulator Independence” test were analyzed for power consumption.

All runs that produced candidate gates resulted in candidate gates that were simulator independent, meaning that they ran on both HSPICE and SPECTRE without failing and with the correct logical output. Of these, 80% produced candidate gates that were able to pass tests on input order. None of the candidate gates produced by any run passed the library dependence test or the time test, which was an anticipated outcome. However, these tests may be safely ignored for the purposes of this project. The library test may be ignored because we are not looking to directly analyze the gates, therefore a gate that only works under the processes described by the fabricator in their library file is acceptable, since that is the behavior expected after fabrication. It should also be noted that the timescale tests changed the simulation from measuring on the millisecond timescale to the nanosecond timescale. Because the circuits are expected to take longer to reach a steady state due to subthreshold operation, the results of the time test will also be ignored. The time test was preserved in the format used to allow a more direct comparison to previous research, and if this method was to be used for fabrication, the test should be altered to test a longer timescale. This could be done by converting all times to 10x the original time, to allow for a more useful test.

### 6.3.2 Power Analysis

HSPICE was used to analyze the power use of each candidate circuit that passed categorization with both simulator and input independence. Each possible power supply, the inputs, as well as $V_{DD}$ and ground, were monitored for power use similarly to [14]. This allows for an accurate method of simulating power usage for the candidate gates. A traditional NAND gate was also simulated and analyzed in this way to be reported as a control.

With each passing candidate gate analyzed for power, the best performing gate was selected from each valid run. The average switching and steady state power usage is given. These results were measured based on switching every 5ms and averaging the power use across the entire time of both the switch and the stable state. These results are given in table 6.1. In this table, the most efficient column shows an average of the most efficient gate in each run. It is important to note that these power results are based on the average current and the set $V_{DD}$ across the time required to switch or the 5ms steady time.

The standard CMOS gate provides much better power results, which is expected given the presence of structures such as the input-source. It is also important to note that this is the average gate generated across the runs. The most power efficient gate generated was orders of magnitudes more efficient, but it was still less power efficient than a standard CMOS NAND gate. The most efficient gate generated also has worse power consumption than the standard NAND gate, but it is much closer to that target. Because of this, care should be taken when using these gates to ensure power usage can still meet requirements.

Table 6.1: High Voltage Subthreshold Power Results

| State | Candidates (W) | Most Efficient (W) | Reference NAND Gate (W) |
|---|---|---|---|
| Steady @ .5V | 8.8e-10 | 9.0e-13 | 7.1e-13 |
| Switching @ .5V | 5.8e-10 | 4.9e-11 | 1.2e-11 |
| Steady @ 3.3V | 0.00050 | 5.6e-11 | 4.7e-12 |
| Switching @ 3.3V | 0.00047 | 7.8e-9 | 7.9e-11 |

## 6.4 Low-Voltage Subthreshold and Multi-Polymorphism

Low-voltage subthreshold polymorphism is polymorphism that is present across the subthreshold line, but still at a voltage suitable for low power design. In this case, the $V_{DD}$s used by the evolutionary algorithm was .4V, .7V, and 1.0V. This allows the evolutionary algorithm to search for not just low-voltage subthreshold polymorphism, but also subthreshold multi-polymorphism.

A second change was also made to the evolutionary algorithm for this search: structure seeding. In this case, the structure that was seeded was the input-source structure. This structure is defined as a path through transistor source and drains from an input to the output. The evolutionary algorithm was run in two batches, one with input-source seeding and one without to allow comparisons to be drawn between the two. Each evolutionary algorithm batch contained 45 runs, for a total of 90.

Similar to the high-voltage subthreshold runs, each candidate circuit found by the evolutionary algorithm was put through categorization. The categorization results are presented in 6.4.1. The circuits that passed the categorization were selected for power analysis in 6.4.2. Multi polymorphism is discussed in 6.4.3.

Table 6.2: Low-Voltage Categorization Results

| Category | Standard Run Count | Seeded Run Count |
|---|---|---|
| No Candidates | 25 | 25 |
| Valid Candidate Found | 20 | 20 |
| Simulator Independence | 10 | 14 |
| Input Independent | 4 | 8 |
| Library Independent | 0 | 1 |
| Time Independence | 0 | 0 |

### 6.4.1 Categorization

The evolutionary algorithm was executed for 90 runs, 45 with input-source seeding and 45 with no seeding. Similar to the high-voltage subthreshold runs, it is best to analyze these on a per-run level rather than an overall per-gate level, as well as compare between the input-source seeded algorithm and the standard algorithm. A table showing the results is given in 6.2.

The overall results were positive, with 40 of the 90 runs finding at least one valid low-voltage subthreshold polymorphic circuit. Of these 40 valid runs, 24 found candidate circuits that are independent of the simulator used, and 12 runs had input independence. As mentioned in the high-voltage subthreshold categorization section, 6.3.1, library and time independence are not strictly required in this context. It is interesting to note that only 1 run from the seeded evolutionary algorithm has a candidate that are library independent.

The seeded batch performed notably better than the unseeded batch. With input-source seeding, there was a 40% increase in runs with simulator independence and a 100% increase in runs with input independence. The seeded batch also produced one run with library independence, while the unseeded and high-voltage evolutionary algorithm did not produce any. An

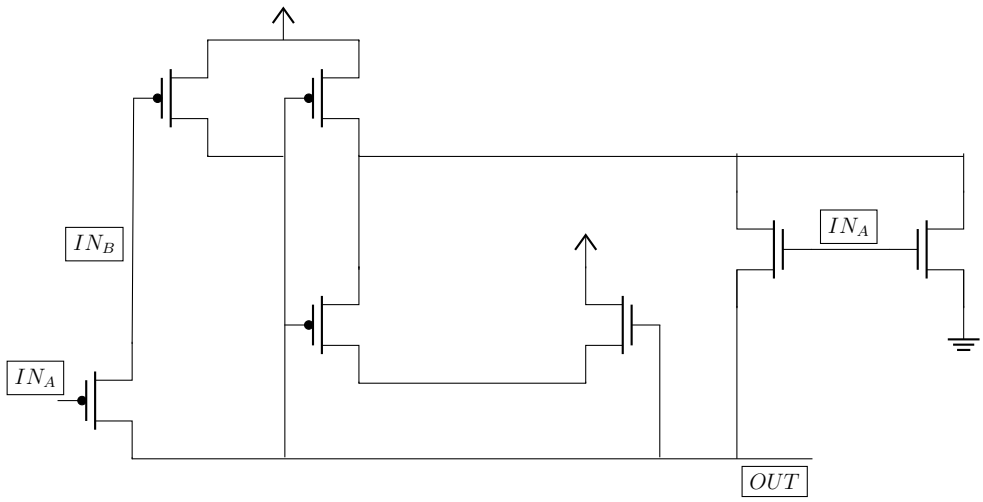



Figure 6.3: This figure shows a polymorphic NOR/NAND circuit generated by the input-source seeded evolutionary algorithm.

example of a gate generated by the seeded algorithm is given in figure 6.3.

### 6.4.2 Power Analysis

The power use and analysis is similar to that of the high-voltage subthreshold gates, with the addition of a third voltage state. The circuits analyzed for power were all generated by the evolutionary algorithm, either in the standard form or seeded with an input-source structure. Table 6.3 shows the results of this power analysis, split between the standard, seeded, and best gates, and compared with the standard NAND and AND CMOS circuit. The gate selected as BEST exhibited AND/XNOR polymorphism and was selected as the gate with the least power usage across the entire simulation.

Overall, the results of this power analysis are better than the high-voltage gates, but they still fall short of the CMOS NAND gate in some categories. This is not surprising for the same reason that higher power usage is not surprising in the high-voltage case. The gate used as BEST did perform significantly better than the reference CMOS AND gate in multiple switching

Table 6.3: Low Voltage Subthreshold Power Results

| State | Standard (W) | Seeded (W) | Best (W) | NAND (W) | AND (W) |
|---|---|---|---|---|---|
| Avg Stable @ .4V | 5.5e-10 | 6.9e-11 | 2.5e-12 | 4.1e-13 | 6.97e-13 |
| Avg Switch @ .4V | 1.1e-11 | 4.8e-11 | 1.7e-12 | 6.6e-12 | 9.81e-12 |
| Avg Stable @ .7V | 9.7e-11 | 1.2e-10 | 4.5e-12 | 1.2e-12 | 4.0e-12 |
| Avg Switch @ .7V | 1.9e-10 | 8.5e-11 | 3.0e-12 | 3.7e-11 | 9.11e-11 |
| Avg Stable @ 1.0V | 1.4e-10 | 1.7e-10 | 6.4e-12 | 2.6e-12 | 5.0e-12 |
| Avg Switch @ 1.0V | 2.8e-10 | 1.2e-10 | 4.3e-12 | 1.78e-10 | 3.8e-10 |

categories. This could be due to the method used for selecting the gate. One thing to note is that power usage was not a factor in the evolutionary algorithm fitness function, and the addition of this criteria to the fitness function could help lower the average power usage. Further investigation of this is required and is included in the future work chapter at 7.3.

### 6.4.3 Multi-Polymorphism

Multi-polymorphism is defined in 5.3.2 as a single polymorphic circuit that has more than two functions, or at least two transition points. These points can be using the same environmental condition, such as $V_{DD}$, or they can be different, such as $V_{DD}$ and sample rate. The low-voltage subthreshold evolutionary algorithm was configured so that it could search for multi-polymorphism simultaneously with single polymorphism.

The algorithm can detect multi-polymorphism across three established $V_{DD}$ voltage levels. One gate that was discovered by the algorithm is presented in figure 6.4 and further reduced to the circuit presented in figure 6.5, by removing the transistors that are not impactful to the output according to the simulators. The removal of these transistors was validated by rerunning the categorization algorithm and validating that the tests that passed with the initial circuit also still

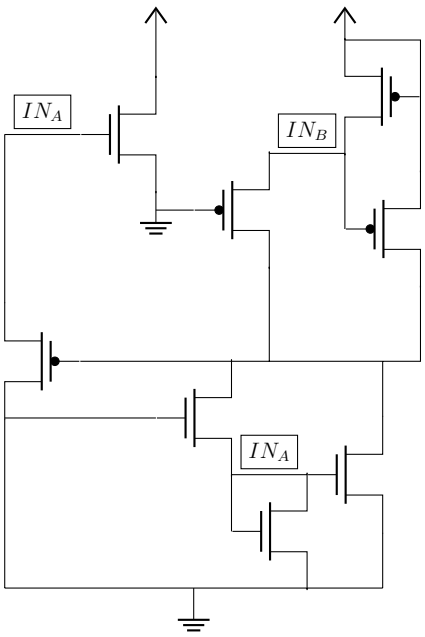



Figure 6.4: This is an example of a low-voltage subthreshold circuit, that also has multi-polymorphism. This shows the circuit as discovered by the evolutionary algorithm.

passed with the modified one. This circuit is also an example of a circuit that does not make intuitive sense, since the output cannot be readily determined with an input of A=0 and B=1, since all transistors will be off.

This circuit passed the simulator independence tests and the input order independence tests, maintaining multi-polymorphic functionality. At .4V it is NOR, .7V it is $B_{INV}$, and at 1.0V it is NAND. This is shown in the functional chart, figure 6.6. This is one example of multi-polymorphism generated by the evolutionary algorithm, and there are relatively few other examples, including [1]. Subthreshold multi-polymorphism is able to be generated by the evolutionary algorithm and does pass categorization.

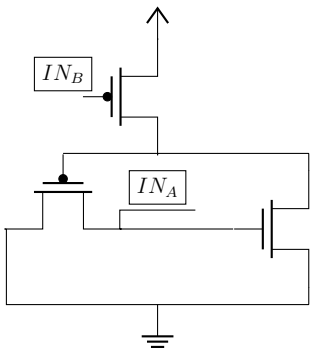



Figure 6.5: This figure is the same circuit as figure 6.4, but this version has unnecessary transistors removed. This circuit passes the same categorization tests as before.

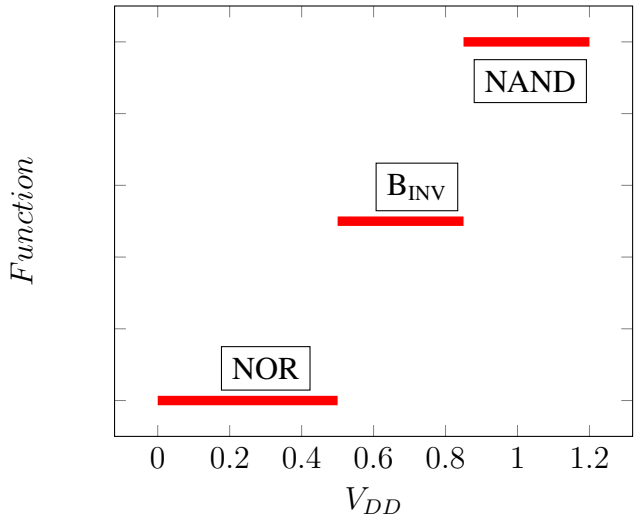



Figure 6.6: This figure shows the functional chart for the circuit shown in figure 6.5

### 6.4.4 Seeded and Non-seeded Algorithm Comparison

One primary contribution of this chapter is the use of the input-source structure from chapter 4 and [49]. The evolutionary algorithm was modified to reject any randomly generated gate to possess an input-source structure, where there is a path through any number of transistors that can allow an input to directly drive an output. It is important to note that this structure was not required to be maintained during the algorithm, so it can be mutated away. A comparison between the two algorithms on speed and number of candidate circuits is presented in table 6.4 and 6.5.

Table 6.4: Seeded and Unseeded Average Earliest Generation Found by Category

| Category | Unseeded | Seeded |
|---|---|---|
| Valid Candidate Found | 29.9 | 17.0 |
| Simulator Independence | 33.3 | 16.6 |
| Input Independence | 28.8 | 26.4 |
| Library Independence | NONE | 48 |
| Time Independence | NONE | NONE |

Table 6.5: Seeded and Unseeded Total Candidates Found by Category

| Category | Unseeded | Seeded |
|---|---|---|
| Valid Candidate Found | 27,114 | 50,825 |
| Simulator Independence | 10,150 | 7,642 |
| Input Independence | 1,903 | 3,687 |
| Library Independence | NONE | 1 |
| Time Independence | NONE | NONE |

Table 6.4 shows the average earliest generation that a candidate was found in a given run. Table 6.5 shows the total number of candidates found by maximum category passed. The candidates are included only once in the table, so a candidate that passed the input independent test is not included in the simulator independent count.

Table 6.4 is used to show the improvement in overall speed. The table shows that on average in each category, for when a candidate is found, the seeded algorithm finds it in an earlier generation. The total number and results of the runs are given in table 6.2. Despite the small sample size, the results do show an improvement in the algorithm's speed.

Table 6.5 shows the total number of candidate circuits discovered. This table indicates that overall, more candidates were found with the seeded algorithm, and while a majority of the improvement was seen in the simulator dependent category, a significant improvement was also

Table 6.6: Theoretical Watermarked Circuit Power Differential

| **Replaced** | **Stable @.4V** | **Switching @.4V** | **Stable @.7V** | **Switching @.7V** |
|---|---|---|---|---|
| 0.1% | 100.26% | 99.92% | 100.01% | 99.9% |
| 1% | 102.6% | 99.17% | 100.13% | 99.03% |
| 5% | 112.93% | 95.87% | 100.63% | 95.16% |
| 10% | 125.87% | 91.73% | 101.25% | 90.33% |

observed in the input independent category. A single candidate that was Library independent was also observed in the seeded run.

## 6.5 Subthreshold Polymorphic Circuit Watermarking

Circuit watermarking using polymorphism is shown in [4] and [18]. In a polymorphic circuit watermarked design, a subset of the total gates are replaced with polymorphic circuits. Using the power analysis information for the best case seeded design in table 6.3, the power use of a watermarked circuit with this gate, replacing CMOS AND gates at .4V and .7V, can be approximated. The results of this are shown in table 6.6.

This theoretical analysis shows that the power usage of a watermarked circuit with a small percentage of transistors replaced can yield a minimal increase in power consumption. If the standard operating voltage is .7V, making a substitution of 5% of transistors will only yield an increase of .63% during the steady state of circuit operation, and can result in a decrease in the switching current used. This application is in line with the percentages presented in [18], and the results show that this can be applied in the subthreshold domain.

## 6.6 Stuck-at Fault Detection with Polymorphic Gates

Polymorphic circuits can be used for fault detection, as shown in [15]. Subthreshold fault detection is particularly important due to reliability issues in subthreshold, and subthreshold fault detection is shown in [30]. This section takes the ideas presented in these sources and investigates polymorphic fault detection in the subthreshold domain. The target structure for fault detection in this initial demonstration is the D-Flip-Flop and later the C17 benchmark circuit. The fault model that is discussed is the stuck-at fault model, where a given gate input is stuck-at a particular logic level, either high or low.

### 6.6.1 Example using a D Flip Flop

#### D-Flip-Flop Structure and Detecting Faults

The D-Flip-Flop is an important digital logic structure, operating essentially as a single bit of memory. It is a rising-edge triggered flip-flop, meaning its output changes value only when the clock/enable signal transitions from low to high. The simple design of the D-Flip-Flop makes it ideal for demonstrating one of the potential applications for subthreshold polymorphic fault correction in practice.

Table 6.7: NAND D-Flip-Flop Truth Table

| **D** | **Clock/Enable** | **Q** | **$Q_{BAR}$** |
|---|---|---|---|
| 0 | 0 | Q | $Q_{BAR}$ |
| 0 | 1 | 0 | 0 |
| 1 | 0 | Q | $Q_{BAR}$ |
| 1 | 1 | 1 | 1 |

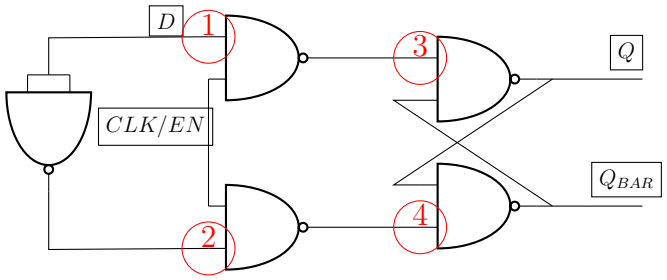



Figure 6.7: This figure shows the D-Flip-Flop gate level diagram with the target fault locations circled.

The D-Flip-Flop takes two inputs: a data input (D) and a clock/enable signal. It produces an output (Q). The standard NAND-based D-Flip-Flop copies the data input (D) to the output (Q) when the clock/enable signal is high. When the clock/enable signal is low, the data input is ignored, and the output remains stable. Table 6.7 shows the truth table for a standard NAND D-Flip-Flop.

In this section, we explore four potential locations for circuit defects (stuck-at faults) in the D-Flip-Flop circuit. These locations are circled in the diagram of Figure 6.7. They correspond to the inputs of each of the four NAND gates used in the circuit.

The standard D-Flip-Flop, built with NAND gates, can detect some circuit defects without the need to store the correct answers to the test patterns. A defect is considered detectable under this system if, for some valid single input value, the output can be shown to be in an invalid state. These detectable defects include the data input signal (D) being stuck-at-1, the inverted data signal being stuck-at-0, and the middle lines being stuck-at-0. This scenario results in both the output (Q) and its complement ($Q_{BAR}$) being the same, which is discussed further below.

Table 6.8: D-Flip-Flop Stuck-At Table

| **Fault** | **Mode** | **D** | **Clock/Enable** | **Q** | **$Q_{BAR}$** |
|---|---|---|---|---|---|
| 1 Stuck Low | NOR | 1 | 0 | 0 | 0 |
| 1 Stuck High | NAND | 0 | 1 | 1 | 1 |
| 2 Stuck Low | NAND | 0 | 1 | 0 | 0 |
| 2 Stuck High | NOR | 1 | 0 | 1 | 1 |
| 3 Stuck Low | NAND | 0 | 1 | 1 | 1 |
| 3 Stuck High | NOR | 1 | 0 | 0 | 0 |
| 4 Stuck Low | NAND | 1 | 1 | 1 | 1 |
| 4 Stuck High | NOR | 0 | 0 | 0 | 0 |

## Fault Detection Using Standard NAND Gates

Stuck-at fault detection using standard methods require that multiple test patterns be generated and the output checked for correctness. This approach requires the correct outputs to be stored and recognizable in order to detect every stuck-at fault possible in the system.

This requirement can be mitigated in a situation such as the D-flip-flop circuit since the two outputs are know to always have opposite values, which gives a simple test for incorrect behavior: the outputs are the same. Because of this, half of the faults can be tested by simple observing the values on the outputs. The faults that can be detected are shown in table 6.8, under the NAND mode. The exact input patterns required to detect the NAND detectable faults are shown in this table as well. Introducing polymorphic gates allows all identified faults to be checked by only checking if the outputs are valid, as detailed in the next section.

Table 6.9: NOR D-Flip-Flop Truth Table

| **D** | **Clock/Enable** | **Q** | **$Q_{BAR}$** |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | Q | $Q_{BAR}$ |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | $Q_{BAR}$ |

## Polymorphic D Flip Flop Fault Detection

The method used here to detect faults is simply to check for logical inconsistency between the outputs. In the case of the D flip flop, this happens when the Q and $Q_{BAR}$ outputs are logically the same value. This allows for simple fault detection logic, since an XOR gate can be places with both inputs set as each output, and if the result is high a fault is present. This removes the need to store the expected result of the test patterns for comparison.

The stuck-at faults that can be detected by the standard NAND gate in this form are incomplete, as shown in table 6.8 under the NAND mode. If the NAND gates are substituted for polymorphic NAND/NOR gates, a much larger range of stuck-at faults can be tested without the need for any additional inputs or outputs beyond the XOR gate previously mentioned, allowing the complete table 6.8 to be tested.

In the general sense, the NOR D-Flip-Flop is an inverse of the NAND D-Flip-Flop. Where the NAND structure is triggered by a rising edge, when the clock/enable bit is high, the NOR structure is conversely triggered by a low clock/enable bit. This is shown in the NOR D-Flip-Flop truth table shown in table 6.9.

By implementing polymorphic NAND/NOR gates, the D-Flip-Flop can be swapped from NAND to NOR, allowing for full coverage of the types of detectable faults. When operating as

a NOR gate, the output will show identical values for both Q and $Q_{BAR}$ as an indicator of a fault. When the input should be 1, the clock is held low, and the middle lines are stuck at 1, the NOR gate will indicate a fault if the D input is stuck at 0 or if the D inverse input is stuck at 1. A full table showing each state of the stuck-at faults is presented in table 6.8.

Polymorphic NAND/NOR gates can detect a wider range of faults using this system compared to using only NAND or NOR gates in a D flip-flop, as shown by their complementary rows in the stuck-at fault table in table 6.8. This allows for more comprehensive fault detection during circuit testing. To ensure the functionality of the evolved gates, HSPICE simulations were performed to verify that they exhibited the desired subthreshold polymorphism with NAND/NOR functionality. Gates that met these criteria were then used to replace traditional NAND gates in a simulated D flip-flop. These simulations, using both circuits derived from the standard evolutionary algorithm and the input-source seeded version, were performed using HSPICE to confirm their functionality.

### 6.6.2 Enhancing Stuck-at Fault Testing with Polymorphic Gates

The D flip flop example previously described required every gate to be replaced with a polymorphic gate. This is undesirable due to some of the overhead involved with most polymorphic gates (increased power use and space), which is described more in section 6.4. In this system, only a subset of gates are replaced in an effort to reduce the number of test patterns required to detect a single stuck-at fault in the system. The general flow of this algorithm is given in 6.8 and involves selecting test patterns using traditional methods to cover a majority of the test cases, or all of the simple cases in situations where there are hard to isolate faults. Once these are

identified, examine the remaining cases and determine what polymorphic gates can be used to replace traditional gates to allow for the targeted faults to be detected in the altered circuit. The overall goal of this algorithm is to balance the substitution of polymorphic gates with the decrease in time that it would take to run the tests on the circuit after reducing the required pattern count. This algorithm can be shown below using the C17 benchmark circuit.

## Case Study: Benchmark Circuit C17 [55]

The C17 benchmark circuit, containing only 6 NAND gates, serves as a simple example to demonstrate the algorithm's effectiveness. This algorithm aims to replace the minimum number of gates with polymorphic gates to reduce the required test patterns for fault detection.

The circuit layout is shown in Figure 6.9. It has 5 inputs and 2 outputs. Detecting all single stuck-at faults on every gate's input requires 4 test patterns. The initial steps of the algorithm identified two candidate test patterns (in order of $IN_0$ to $IN_5$): 00101 and 11111. These patterns allow for identification of 17 out of the 24 total stuck-at faults under consideration, as detailed in Table 6.10.

The next step involves examining what type of polymorphism might be most effective based on the faults not yet detected. In this instance, XOR/NAND polymorphism was chosen due to its property of having an equal split of logic levels at the output, making it easier to detect changes caused by stuck-at faults. Consequently, the final two gates, $N_4$ and $N_5$ in Figure 6.9, were selected for replacement with XOR/NAND gates. This allows any change in the inputs to both output gates to be reflected in the final output.

With this change, a single additional pattern (11010) was identified and selected. When

Table 6.10: C17 Benchmark Fault Detection Patterns

| **Location** | **00101** | **01010** | **10000** | **11111** | | **11010 *** |
|---|---|---|---|---|---|---|
| $N_0$ Top | 1 | | | 0 | | |
| $N_0$ Bottom | | | 1 | 0 | | 1 |
| $N_1$ Top | | 1 | | 0 | | 1 |
| $N_1$ Bottom | 1 | | | 0 | | |
| $N_2$ Top | 1 | 0 | 1 | | | 0 |
| $N_2$ Bottom | | 0 | | 1 | | 0 |
| $N_3$ Top | 0 | | | 1 | | |
| $N_3$ Bottom | 0 | 1 | 1 | | | 1 |
| $N_4$ Top | 0 | | 0 | 1 | | 0 |
| $N_4$ Bottom | 0 | 1 | 0 | | | 1 |
| $N_5$ Top | | 1 | 0 | 0 | | 1 |
| $N_5$ Bottom | 1 | | 0 | 0 | | 0 |

considering all fault locations, these three patterns (00101, 11111, and 11010) can detect all 24 cases. If no replacement is made, four patterns are required (with 01010 and 10000 added to the initial two). Table 6.10 summarizes the total patterns required, with the starred column indicating XOR mode and the others representing NAND mode.

## 6.7 Discussion

In conclusion, polymorphism can be brought to the subthreshold domain. Much care must be taken to ensure that the results are as expected, and that no issues arise, but simulation and categorization both show that it is possible. This chapter looked at both high-voltage and low-voltage subthreshold polymorphism, seeding the evolutionary algorithm, a power analysis of each, and a possible application of subthreshold polymorphism.

The effect of seeding an evolutionary algorithm with desired or expected structures was also positive. This created both a higher likelihood of a given run finding acceptable gates, decreased the time needed for finding gates, and it increased the overall categorization results by finding some gates that are library independent. This is a possible avenue of continued research, and it should be explored further.

Power analysis was conducted on the resulting candidate gates. On average, the high-voltage gates performed much worse than a control CMOS NAND gate, but the most efficient versions performed more closely to the CMOS gate. Adjustments might be possible to solve this issue, and are mentioned in the future work chapter, 7. The power analysis of low-voltage gates was closer to the CMOS gate. The most efficient version of the low-voltage gate was still less efficient in some cases, but only by about one order of magnitude. The loss of power efficiency compared to the standard is minor enough that the additional functionality makes this potentially a valuable consideration in applications.

One such function is fault checking. This chapter explored one possible setup, where a D-Flip-Flop could be checked more completely for stuck-at faults, if the NAND gates were replaced by polymorphic NAND/NOR gates. Simulations confirmed that the evolved gates can be used to simulate a D-Flip-Flop and that the gates performed as expected in both the NAND and NOR mode. A second application discussed was watermarking. Which can alleviate some of the power inefficiency by only replacing a small percentage of the overall gate count.

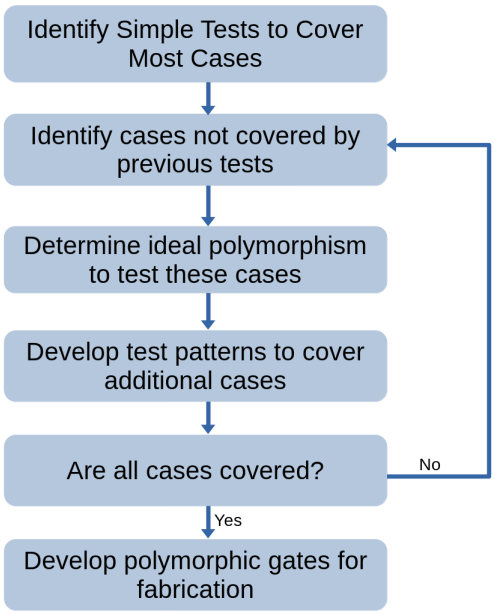



Figure 6.8: This figure shows the basic flow for partial gate replacement to allow for a reduced or more comprehensive test pattern generation for fault detection.

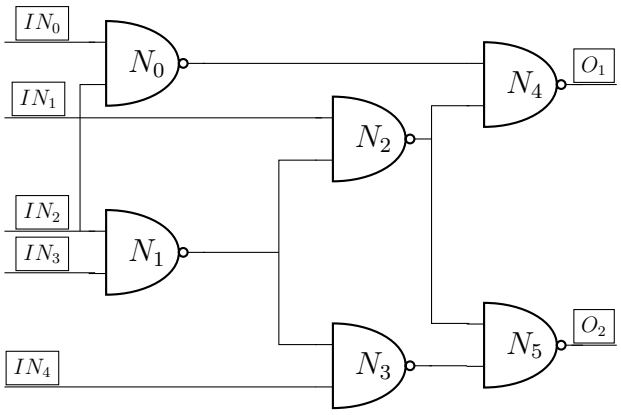



Figure 6.9: This figure is a schematic showing the C17 benchmark circuit.

# Chapter 7: Future Work

## 7.1 Chapter Organization

Much work was presented on polymorphism, but there is still a tremendous amount that is unknown. The crux of this dissertation was a multi-pronged approach to polymorphism and driving the field forward to determine fruitful avenues for future work and discovery. Having discussed polymorphic gates, their evolution, and exciting applications for practical use, we now turn to the most promising future directions for the research presented here. As well as describing avenues of research that have not been explored yet, this chapter will also discuss unexpected results. These outcomes are paired alongside possible theories of explanation and additional avenues for exploring those theories. The machine learning evolutionary algorithm presented the most unknowns and is discussed in-depth in 7.4.

This chapter is organized in categories that follow mostly independent lines of research. Section 7.2 presents possible methods of furthering the categorization of evolved circuits and addresses some of the shortcomings of the current methodology. Section 7.3 focuses on the low power aspect of polymorphic design, particularly with respect to subthreshold, and it presents some ways to address the power concerns brought up in 6.3.2 and 6.4.2. The final future work section is 7.5, which is where additional facets of multi polymorphism are hypothesized and possible experiments are presented. Section 7.6 discusses how the algorithms should be adjusted

to allow for fabrication, as well as the contributions of fabricating this research. Section 7.7 summarizes the above and presents the key take-aways of the polymorphism effort detailed here.

## 7.2 Categorization

Categorization was first presented in this work in section 4.2.2. In the proposed method, a set of four tests are conducted with two simulators each. The four tests are using evolutionary algorithm library, ideal library, input independence, and time independence. The first set of tests, evolutionary algorithm library, is a HSPICE and SPECTRE simulation that is identical to the simulation used during the evolutionary algorithm. This is to act both as a control to ensure that the circuit behaves correctly, and to test if there is simulator dependence, where HSPICE may be creating situations and criteria that are unique to that simulator. Passing this test is a requirement before the following tests are considered.

The second set of tests is to determine if the library used produces polymorphism. The ideal library test is not a requirement for proceeding since library specific circuits are acceptable in instances when the library matches the fabricator that will be used. However, it should be noted that library specific impacts may be more difficult to analyze with the circuit diagram, so the way in which these circuits function may not be immediately obvious. The next test reverses the input order, to ensure that the input order does not impact the presence of polymorphism. The final test changes the timescale used for the simulation, between milliseconds and nanoseconds. This checks for both time-based polymorphism and to ensure that, if this is not desired, that the circuit behaves correctly and has reached a steady state.

The first area of possible improvement on this work is the input independence test. At

the current time, only two input patterns are tested: 00, 01, 10, 11 and the reverse, 11, 10, 01, 00. This could be changed to test all possible input combinations, and only pass the gate if all combinations pass, ensuring more robust gates.

The second area of improvement proposed here that all tests are currently independent. Each test discussed in the preceding text is conducted by two simulators, however, investigations into how these tests may interact were not conducted. As an example, it is possible for both tests to pass independently but to fail to exhibit the same polymorphic behavior upon changing the timescale along with the input. This would indicate the two tests are not independent of each other. If these two tests are not independent of each other, it is possible for both tests to pass independently of each other, but if the timescale is changed and the input order is changed the circuit will fail to exhibit the same polymorphic behavior.

A third area of research is to evaluate the actual robustness of the resulting circuits in silicon. To do this, a number of circuits that pass individual tests or groups of test would need to be fabricated and their robustness would need to be measured by physical equipment. The goal of this would be to prove that the categorization algorithm can correctly select robust gates from those produced by the evolutionary algorithm without needing to go to fabrication to test each one.

## 7.3 Low Power

Power analysis of the evolved polymorphic circuits was presented in sections 6.3.2 and 6.4.2. In these sections, high-voltage and low-voltage subthreshold polymorphic circuits are presented and the power use of the candidates that passed categorization was carried out. The

reported results are the average power consumption of the most power efficient gates analyzed from each run. The results show that the evolved circuits are less power efficient than a standard CMOS NAND gate.

One primary reason for this lower efficiency is that the evolutionary algorithm is building structures that are not standard CMOS, but are initially random connections of transistors. Some of these random connections may yield circuits that are polymorphic but are incredibly power inefficient. Because power efficiency is not part of the fitness model, the algorithm is not currently tuned to evolve power efficient gates. The primary line of research to further low power polymorphism would be to include a power analysis component in the fitness function.

The fitness function is a very important part of the evolutionary algorithm. It determines which circuits pass on to the next generation, and it ultimately determines the quality of each circuit that is simulated. It is also the reason that the categorization step is important, because the fitness function can cause the evolutionary algorithm to find results that fit the fitness function exactly, but that are not actually polymorphic. This particular issue is presented in [3] as one problem with evolutionary algorithms, and is the problem that categorization attempts to solve. The second consideration is how to weight the power usage within a given fitness function. If the power analysis is weighted equal to the polymorphism, the evolutionary algorithm might generate a circuit that is refined to be extremely power efficient, but does not contain robust polymorphism. If the polymorphism is weighted much higher than the power usage, a highly robust polymorphic circuit might be weighted above an acceptably robust polymorphic circuit that has better power consumption. One possible solution to this is a multi-tiered fitness function.

In a multi-tiered fitness function, a threshold is set on the first function where all circuits that pass that threshold are considered acceptable to continue to the next phase. The passing

circuits then are subjected to the second fitness function, which will rank them based on a second criteria, such as power analysis. This function would pass all circuits that passed the first function, ranked by the second function, then it would rank the remaining circuits that did not hit the threshold. However, this does not completely resolve the problem with the fitness function itself. When power usage is added to the fitness function, a new test to categorization must also be included. This test would ideally employ another simulator and another method of power analysis to validate that the circuit is considered power efficient across different simulators.

## 7.4 Machine Learning Evolutionary Algorithm

One research avenue pursued was to incorporate some machine learning to the evolutionary algorithm. The chosen method for this was to add a gradient descent algorithm that would run after the primary fitness function completed and a set threshold was met, similar to the multi-tiered fitness function in 7.3, but the second ”tier” of fitness function was a gradient descent algorithm operating on the width and length parameters of the polymorphic gates. The goal of this approach was to increase the robustness compared to the number of total simulations needed for the algorithm.

For this algorithm, the gates were evolved and the most robust gates in a generation that exhibited some polymorphism were then passed to the gradient descent algorithm to optimize the gates to a more robust state. The evolutionary algorithm was run with the same random seed twice, once with the gradient descent and once just the evolutionary algorithm. Consistently, the evolutionary algorithm performed with the same robustness as the evolutionary algorithm plus gradient descent, but with fewer simulations. The gradient descent algorithm did improve

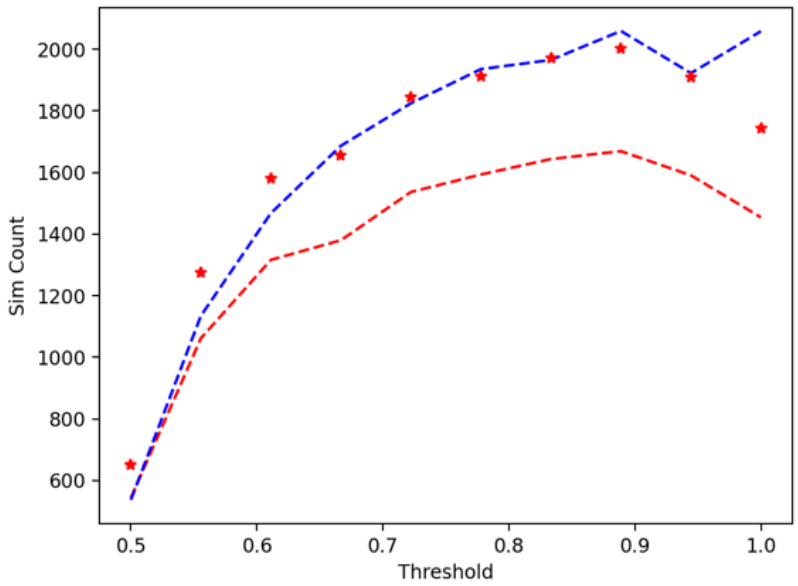



Figure 7.1: This figure shows the number of simulations VS the resulting threshold for gradient descent vs just evolutionary algorithm. In this figure, the blue line on top is gradient descent, the red line on bottom is the standard evolutionary algorithm, and the red stars mark close to the top line mark the spots 20% above the evolutionary algorithm, which was set as the cap on the gradient descent simulation count.

the robustness slightly, but overall the improvements were not enough to justify the increase in simulation count. Figure 7.1 shows these results, as observed. The chart shows the threshold, which is the robustness of the gate as measured by closeness to the ideal voltage levels (an output of 2.8V with $V_{DD}$ = 3.3 would have a threshold value of .85 since 2.8 is 85% of 3.3). This shows that the gradient descent algorithm did not improve on the threshold with respect to the simulation count, and it performed worse overall.

One possible reason for this is that there are many local minimums in the threshold gradient. This would mean that small mutations in the structure would send the circuit into a new region of the surface and that the gradient descent to find the local minimum is not as effective as searching for new spots that are close, but that possess differing structures. If this is the case, a gradient descent on the width and length values will not ever consistently improve the threshold

vs simulation counts.

There are two potential research opportunities emerging from this result: one to test the above theory that the gradient descent on width and length values will not be consistently usable to improve the result and the other being to implement a machine learning algorithm that will change the actual structure of the gates.

In order to test the width and length gradient descent algorithm viability, an algorithm could be built that will test the actual improvement for a given circuit that is possible by altering the width and length values. This value could be compared to the improvements possible by altering one or two of the structure elements. If this is done for enough randomly selected polymorphic circuits, a pattern showing that structure alterations are better than width and length alterations may emerge. This would be evidence to point to the theory that the gradient is just more flat on the width and length as opposed to the structure gradient.

A structure altering algorithm would be able to possibly replace the evolutionary algorithm altogether. One research avenue would be to use a vision based model that logically can ”see” the circuit and relations and can make a change to the model schematic. This model could then use the fitness function from the evolutionary algorithm to determine a score for the resulting circuit.

## 7.5 Multi-Polymorphism

Multi-polymorphism is defined as the existence of multiple points where the circuit changes function. This can be manifested in a number of ways, one of which is the presence of three or more polymorphic functions all controlled by the same external variable, such as VDD. Another example would be if two or more functions are controlled by two or more external variables, such

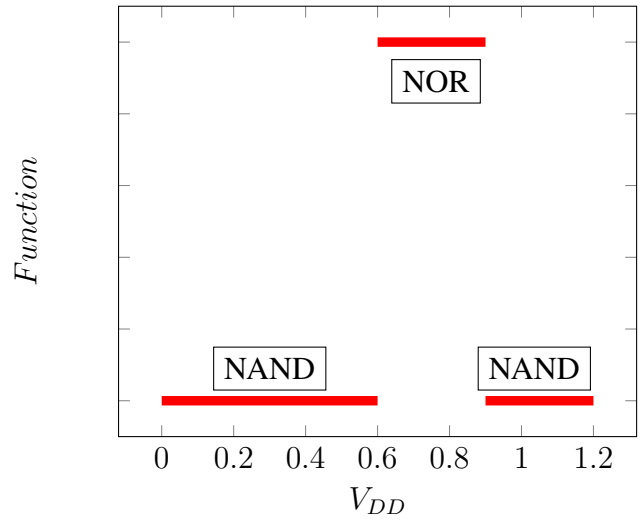



Figure 7.2: This figure the function chart for a multi-polymorphic circuit with only two functions, but one is only visible from .6V $<V_{DD}<$.9V.

as a gate that only exhibits polymorphism at both high temperatures and high VDD. Additionally, a polymorphic gate that exhibits two functions but switches back to the original function again, such as a gate that oscillates between NAND and NOR through two inflection points would also qualify as polymorphic. The switching between NAND and NOR functionality is shown in figure 7.2.

Future research on multi-polymorphism could take on a couple of different forms: changes to the evolutionary algorithm to seek out this functionality, and focusing on possible applications of this technology. To change the evolutionary algorithm, the fitness function would need to change to either consider three different functions, or two functions but with a split between one of them. These modifications could be made fairly simply, but the categorization would then need to be adjusted to account for this change.

Possible applications for this technology exist in the hardware security space. If the circuit that has the function chart shown in figure 7.2 is used, a circuit can either only be run at certain voltages, or if it is run at those voltages then a hidden function or watermark can be inserted. If a

circuit that has multiple different external factors that must line up for the polymorphic behavior to be revealed is used, then the parent circuit could be functionally encrypted. The environmental parameters would need to be tuned precisely in order for the circuit to appropriately run, which would result in excellent hardware security.

## 7.6 Circuit Fabrication

Circuit fabrication is the ultimate goal of design and polymorphic circuits have been fabricated in the past [56]. Fabrication of circuits designed using the evolutionary algorithm and categorization described in this dissertation has not been completed, but is the goal of the research presented within this dissertation. To achieve this aim, some adjustments are necessary to the algorithms.

The evolutionary algorithm used for this research would need to be modified to accept the proper library for the target functionality. The parameters used for the evolutionary algorithm, target functions and voltages, would need to be adjusted per the specifications of the fabrication facility. For testing, the algorithm was configured to cap at 50 generations and then start a new iteration from randomly generated gates. This behavior can be modified to allow the algorithm to continue for additional generations, until a valid candidate is found. These modifications are minor compared to the complexity of the algorithm.

The categorization methodology would need to be adjusted as well. Since fabrication would target a particular library, the library independence tests are less critical. The time scale test would need to be adjusted to be realistic time scales for the application of the circuit being designed. The other tests, particularly the simulator dependence test, are hypothesized to be the biggest factors

in categorization.Categorization is an important contribution of this work that can be confirmed in fabrication.

The gates fabricated would be ones from multiple different passing levels of categorization, with multiple combinations of the sources discovered. The contribution of the fabrication would be a confirmation or a rebuttal of the methodology shown here: evolutionary algorithm gives candidates, the candidates are categorized to determine which are the most robust and the most robust are the better candidates for analysis and fabrication. Fabricating this research will show if this methodology can be used to design, via the evolutionary algorithm, circuits that have a higher level of confidence that they will work after fabrication.

## 7.7 Chapter Summary

In conclusion, multiple possible research projects can be continued from this work. Possible options include modifications to the evolutionary algorithm, as with the multi-polymorphism, machine learning, or low power sections. Another project could be to improve on the categorization element of the circuit. Each potential research avenue was detailed and the usefulness described.

The categorization projects discussed include changing the algorithm to contain more comprehensive testing to avoid missed cases. The second project includes combining multiple tests together to test the dependence of one test on the other. The final follow-on project presented is to fabricate the resulting circuits to prove that the categorization algorithm does select the most robust gates from those evolved from the evolutionary algorithm.

The low power research possibilities mainly focus on altering the evolutionary algorithm

to include the low power analysis in the fitness function. There are many considerations before this can be implemented, and these are presented independently, as a multi-tiered fitness function along with adjustments and considerations involving the categorization algorithm. Each of these would need to be addressed before the evolutionary algorithm can actively seek out a more power efficient polymorphic gate.

Embedding machine learning into the evolutionary algorithm was also discussed and attempted. Little improvement was seen and this was unexpected. One possible reason for this is that the width and length functions are not as useful in finding a more robust gate as structure changes, and it is from this initial theory that additional investigation suggestions are built. In the first presented follow-on project, the width and length changes are compared to structure changes to directly see which ones yield the best results per simulation. The second project is to use a machine learning algorithm rather than gradient descent to actually alter the structure of the circuit to see if a better algorithm can be discovered outside of the evolutionary algorithm.

The next section of future research presented involved multi-polymorphism. Two different directions for follow-on research are presented: evolutionary algorithm changes and application research. For algorithm changes, the fitness function would be updated to actively enforce the desire for multi-polymorphism, and the categorization algorithm would be updated to reflect this. The second direction pertains to hardware security application research. Possible applications include circuit encryption with multiple environmental parameters as well as embedded signatures and hidden functionality.

The final section presented fabrication. Fabrication is the primary target of circuit design, and fabricating this research will show that the methodology presented is sound and can be used to design circuits with higher confidence than just with the evolutionary algorithm.

# Chapter 8: Conclusion

In this dissertation, four primary research topics were presented relating to polymorphic circuits. These topics include presenting a framework for converting existing polymorphic circuits into new polymorphic circuits, finding possible sources of polymorphism in circuits, time based and multi-polymorphism, and taking polymorphic gates to subthreshold voltages. Each topic was presented in its respective chapters.

## 8.1 Polymorphic Framework

The polymorphic circuit interoperability framework was created to show that polymorphism itself is possible to separate from a polymorphic circuit. This framework is similar to polymorphic multiplexing, as presented by [5], but it is also distinct because it takes existing polymorphic gates and makes them into a component of a larger gate.

The framework splits the gate into two distinct parts: the polymorphic source and the switchable gate. The polymorphic source is a component, as small as a single transistor in some cases, that takes no formal input and outputs a logic high or logic low based on what polymorphic function should be expressed. The switchable gate is a gate that can switch its function based on the input provided by the polymorphic source.

## 8.2 Possible Sources of Polymorphism

Polymorphic gates are commonly the result of an evolutionary algorithm [3, 5]. Chapter 4 presented four items of note with evolved polymorphic gates: asymmetry, input-sources, non-standard transistor utilization either as a threshold-drop transistor or as a capacitor), and loops.

Categorization was also presented, in which each circuit that the evolutionary algorithm selected as a potential candidate was put through a series of tests to help determine its robustness. The individual tests included testing a different simulator, a different and more idealized library, a different input pattern, and different sample timing. The goal of categorization to to address flaws in evolutionary algorithms where the eventual target of the gate, which is fabrication in the case of polymorphic gates, is different from the actual fitness function, which is just a single simulation.

In order to discover the structures, multiple evolved gates were analyzed and one of these analyzed gates was presented. Also analyzed was a circuit presented by JPL. The analysis of these gates led to the discovery of the structures, which were then used to hand-craft two more polymorphic gates.

## 8.3 Time Based and Multi-Polyorphism

Time based and multi-polymorphism are two aspects of polymorphic gates that were presented. Time based polymorphism is polymorphism that relies on the sample time of the circuit. Multi polymorphism is polymorphism that contains multiple points in which the gate switches function. Each of these make up an interesting facet of polymorphic design.

With time based polymorphism, the sample rate of the circuit is used to change the function. In the example provided, this is because the gate takes longer to reach a high logic level from a low level on a particular input. The effect of this is that an output is allowed to reach a logic high when the gate is given longer to reach this state but is a logic low when then output is sampled quickly. These gates would need to be reset to a known state in-between cycles where they are read quickly to ensure that they are the correct value.

Multi-polymorphism is the presence of multiple points on a circuit where the functionality changes. Examples of this include the presence of three possible functions, or multiple environmental factors that trigger the polymorphism, or circuit switching from one function to a second and then back to the first function. Each of these have particular possible applications to be considered.

## 8.4 Subthreshold Polymorphism

Polymorphic circuits were discovered using the evolutionary algorithm that were in the subthreshold region. The first group explored were high-voltage subthreshold polymorphic circuits, which are subthreshold circuits with $V_{DD}$ = .5 and 3.3V, switching function between the two. The second group were low-voltage subthreshold circuits with $V_{DD}$ = 0.4V, 0.7V, and 1.0V switching function at least once between the three voltage levels.

For high-voltage polymorphism, the expectation was to discover many more gates than the standard polymorphic evolutionary algorithm, since crossing the threshold boundary was expected to have a larger impact on the gate's function. The actual result was that only a relatively small percentage of runs actually evolved a polymorphic gate. The runs were analyzed for power

efficiency, compared with a CMOS NAND gate, and overall did poorly. Theories for this unanticipated outcome were given, as well as some possible applications, and and possible avenues for leveraging high-voltage polymorphism were presented in the future work section.

Low-voltage subthreshold polymorphism was split into two sections: seeded and standard. Seeded runs included gates that were evolved using a modified version of the evolutionary algorithm that forced all initial circuits to have an input-source structure. Standard gates were evolved with no such constraint. The result of this experiment was a dramatic improvement in the number of runs that found a polymorphic gate that passed categorization as input-independent. The seeded runs produced 8 gates compared to the 4 successful gates generated from the standard runs. Additionally, the seeded algorithm also had one run that discovered library-independent gates. Each of the gates that passed categorization were analyzed for power use, and the most efficient gate per run was averaged together and compared with a CMOS NAND gate. The results were better than the high-voltage subthreshold gate, but the results were still less power efficient than the CMOS gate in most cases. Possible enhancements for this were also presented in the future work section.

# Bibliography


[1] A. Stoica, R. Zebulum, D. Keymeulen and J. Lohn, "On polymorphic circuits and their design using evolutionary algorithms", Proc. of Lasted International Conference on Applied Informatics (AI2002), 2002.

[2] A. Stoica, R. Zebulum and D. Keymeulen, "Polymorphic electronics", International Conference on Evolvable Systems, pp. 291-302, 2001.

[3] A. Stoica, R. S. Zebulum, X. Guo, D. Keymeulen, M. I. Ferguson, and V. Duong, "Taking evolutionary circuit design from experimentation to implementation: Some useful techniques and a silicon demonstration," IEE Proceedings - Computers and Digital Techniques, vol. 151, no. 4, p. 295, 2004.

[4] T. Wang, X. Cui, D. Yu, O. Aramoon, T. Dunlap, G. Qu, X. Cui, "Polymorphic gate based IC watermarking techniques," 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, Korea (South), 2018, pp. 90-96, doi: 10.1109/ASP-DAC.2018.8297288.

[5] A. Crha, R. Ruzicka and V. Simek, "Synthesis Methodology of Polymorphic Circuits Using Polymorphic NAND/NOR Gates," 2015 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim), Cambridge, UK, 2015, pp. 612-617, doi: 10.1109/UK-Sim.2015.82.

[6] T. Dunlap, G. Qu and J. Lai, "A Polymorphic Circuit Interoperability Framework," 2019 IEEE 13th International Conference on ASIC (ASICON), Chongqing, China, 2019, pp. 1-4

[7] A. Suarez, H. Oro, L. Peñaredonda, R. Anacan and M. N. Pangilinan, "Design of a New External Signal Controlled Polymorphic Gates," 2016 7th International Conference on Intelligent Systems, Modeling and Simulation (ISMS), Bangkok, 2016, pp. 413-418.

[8] X. Zhang and W. Luo, "Evolutionary design of polymorphic circuits with the improved evolutionary repair," 2013 IEEE Congress on Evolutionary Computation, Cancun, 2013, pp. 2192-2200

[9] L. Sekanina and Z. Vasicek, "A SAT-based fitness function for evolutionary optimization of polymorphic circuits," 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2012, pp. 715-720

[10] X. Cui, X. Li, M. Zhang and X. Cui, "Design of the RRAM-Based Polymorphic Look-Up Table Scheme," in IEEE Journal of the Electron Devices Society, vol. 7, pp. 949-953, 2019

[11] P. Fiser and V. Simek, "Optimum polymorphic circuits synthesis method," 2018 13th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Taormina, 2018, pp. 1-6

[12] Z. Gajda, L. Sekanina, "On Evolutionary Synthesis of Compact Polymorphic Combinational Circuits," Journal of Multiple-Valued Logic and Soft Computing. Philadelphia: 2011, vol. 17, no. 6, pp. 607-631

[13] S. T. C. Konigsmark, D. Chen and M. D. F. Wong, "PolyPUF: Physically Secure Self-Divergence," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 7, pp. 1053-1066, July 2016

[14] J. Nevoral, V. Šimek and R. Růžička, "Power Consumption Analysis of New Generation of Polymorphic Gates," 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Novi Sad, Serbia, 2020, pp. 1-6

[15] R. Ruzicka, L. Sekanina and R. Prokop, "Physical Demonstration of Polymorphic Self-Checking Circuits," 2008 14th IEEE International On-Line Testing Symposium, Rhodes, Greece, 2008, pp. 31-36, doi: 10.1109/IOLTS.2008.23.

[16] U. Banerjee and A. P. Chandrakasan, "A Low-Power BLS12-381 Pairing Cryptoprocessor for Internet-of-Things Security Applications," in IEEE Solid-State Circuits Letters, vol. 4, pp. 190-193, 2021, doi: 10.1109/LSSC.2021.3124074.

[17] Feng Li and Peng-Yung Woo, "Fault detection method for the subcircuits of a cascade linear circuit," in IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 47, no. 8, pp. 1254-1258, Aug. 2000, doi: 10.1109/81.873881.

[18] J. T. McDonald, Y. C. Kim, T. R. Andel, M. A. Forbes and J. McVicar, "Functional polymorphism for intellectual property protection," 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, 2016, pp. 61-66.

[19] S. Koteshwara, C. H. Kim and K. K. Parhi, "Key-Based Dynamic Functional Obfuscation of Integrated Circuits Using Sequentially Triggered Mode-Based Design," in IEEE Transactions on Information Forensics and Security, vol. 13, no. 1, pp. 79-93, Jan. 2018.

[20] M. Al-Haidary and Q. Nasir, "Physically Unclonable Functions (PUFs): A Systematic Literature Review," 2019 Advances in Science and Engineering Technology International Conferences (ASET), Dubai, United Arab Emirates, 2019, pp. 1-6

[21] S. Lu, C. Lee, P. Noel, S. Adham, T. Wong and J. Chang, "A Reliable, Low-Cost, Low-Energy Physically Unclonable Function Circuit Through Effective Filtering," 2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2019, pp. 1-4.

[22] M. S. Mispan, B. Halak, Z. Chen and M. Zwolinski, "TCO-PUF: A subthreshold physical unclonable function," 2015 11th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), Glasgow, 2015, pp. 105-108.

[23] I. Saha, R. R. Jeldi and R. S. Chakraborty, "Model building attacks on Physically Unclonable Functions using genetic programming," 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), Austin, TX, 2013, pp. 41-44

[24] F. Ganji, S. Amir, S. Tajik, D. Forte and J. Seifert, "Pitfalls in Machine Learning-based Adversary Modeling for Hardware Systems," 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2020, pp. 514-519

[25] B. H. Calhoun, A. Wang, N. Verma and A. Chandrakasan, "Sub-Threshold Design: The Challenges of Minimizing Circuit Energy," ISLPED'06 Proceedings of the 2006 International Symposium on Low Power Electronics and Design, Tegernsee, Germany, 2006, pp. 366-368, doi: 10.1145/1165573.1165661.

[26] A. Ren, B. Yuan and Y. Wang, "Design of high-speed low-power polar BP decoder using emerging technologies," 2016 29th IEEE International System-on-Chip Conference (SOCC), Seattle, WA, USA, 2016, pp. 312-316, doi: 10.1109/SOCC.2016.7905499.

[27] S. Gautam, M. M. Singh, V. Bhadauria and M. J. Siddiqui, "Leakage minimization in full adder by using sub-threshold technique," 2017 International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT), Aligarh, India, 2017, pp. 142-146, doi: 10.1109/MSPCT.2017.8363992.

[28] S. Fisher, A. Teman, D. Vaysman, A. Gertsman, O. Yadid-Pecht and A. Fish, "Digital subthreshold logic design - motivation and challenges," 2008 IEEE 25th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, 2008, pp. 702-706, doi: 10.1109/EEEI.2008.4736624.

[29] T. Hiramoto et al., "Ultra-low power and ultra-low voltage devices and circuits for IoT applications," 2016 IEEE Silicon Nanoelectronics Workshop (SNW), Honolulu, HI, USA, 2016, pp. 146-147, doi: 10.1109/SNW.2016.7578025.

[30] S. Nimara, A. Amaricai and M. Popa, "Sub-threshold CMOS circuits reliability assessment using simulated fault injection based on simulator commands," 2015 IEEE 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics, Timisoara, Romania, 2015, pp. 101-104, doi: 10.1109/SACI.2015.7208179.

[31] V. Stopjaková, M. Kováč, D. Arbet and L. Nagy, "Towards Energy-autonomous Integrated Systems Through Ultra-low Voltage Analog IC Design," 2019 MIXDES - 26th International Conference "Mixed Design of Integrated Circuits and Systems", Rzeszow, Poland, 2019, pp. 38-45, doi: 10.23919/MIXDES.2019.8787196.

[32] M. Chanda, D. Sinha, J. Basak, T. Ganguli and C. K. Sarkar, "Design and analysis of adiabatic complex sequential logic circuits in sub-threshold regime for ultra-low power application," 2016 International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur, India, 2016, pp. 1921-1926, doi: 10.1109/ICCSP.2016.7754506.

[33] H. Rady, H. Hossam, M. S. Saied and H. Mostafa, "Memristor-Based AES Key Generation for Low Power IoT Hardware Security Modules," 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), Dallas, TX, USA, 2019, pp. 231-234, doi: 10.1109/MWSCAS.2019.8885031.

[34] S. K. Mathew et al., "$\mu$ RNG: A 300–950 mV, 323 Gbps/W All-Digital Full-Entropy True Random Number Generator in 14 nm FinFET CMOS," in IEEE Journal of Solid-State Circuits, vol. 51, no. 7, pp. 1695-1704, July 2016, doi: 10.1109/JSSC.2016.2558490.

[35] M. Yasin, T. Tekeste, H. Saleh, B. Mohammad, O. Sinanoglu and M. Ismail, "Ultra-Low Power, Secure IoT Platform for Predicting Cardiovascular Diseases," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 64, no. 9, pp. 2624-2637, Sept. 2017, doi: 10.1109/TCSI.2017.2694968.

[36] X. Xu and Y. Wang, "High Speed True Random Number Generator Based on FPGA," 2016 International Conference on Information Systems Engineering (ICISE), Los Angeles, CA, USA, 2016, pp. 18-21, doi: 10.1109/ICISE.2016.14.

[37] S. Malik, "Counter based approach to intellectual property protection in sequential circuits and comparison with existing approach," 2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA), Mumbai, India, 2014, pp. 48-53, doi: 10.1109/CSCITA.2014.6839233.

[38] J. Nevoral, V. Šimek and R. Ružicka, ”PoLibSi: Path Towards Intrinsically Reconfigurable Components,” 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Greece, 2019, pp. 328-334, doi: 10.1109/DSD.2019.00055.

[39] Y. Bi, K Shamsi, J.S. Yuan, P. E. Gaillardon, G.D. Micheli, X. Yin et al., ”Emerging technology-based design of primitives for hardware security”, ACM Journal on Emerging Technologies in Computing Systems, vol. 13, no. 1, pp. 3, 2016.

[40] S. Rai, A. Rupani, P. Nath and A. Kumar, ”Hardware Watermarking Using Polymorphic Inverter Designs Based On Reconfigurable Nanotechnologies,” 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA, 2019, pp. 663-669.

[41] T. Wang et al., “A Novel Polymorphic Gate Based Circuit Fingerprinting Technique,” in Proceedings of the 2018 on Great Lakes Symposium on VLSI, in GLSVLSI ’18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 141-146.

[42] C. Bernard, W. Bryant, R. Becker and J. Di, ”Design of Asynchronous Polymorphic Logic Gates for Hardware Security,” 2021 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2021, pp. 1-5, doi: 10.1109/HPEC49654.2021.9622835.

[43] T. Dunlap, O. Aramoon, G. Qu, T. Wang, X. Cui, and D. Yu, ”A Novel Circuit Authentication Scheme Based on Partial Polymorphic Gates,” 2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Shanghai, China, 2021, pp. 1-4.

[44] Y. Chen, X. Cui, P. Yang, and G. Qu. ”An Anti-removal-attack Hardware Watermarking Method based on Polymorphic Gates”, International Conference on Computer-Aided Design (ICCAD), 2023.

[45] G. Qu and M. Potkonjak. ”Intellectual property protection in VLSI designs: theory and practice”, Springer Science & Business Media, 2007.

[46] R. C. Jaeger and T. N. Blalock, “COMPLEMENTARY MOS (CMOS) LOGIC DESIGN,” in Microelectronic Circuit Design, McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 2016, pp. 367–406.

[47] G. Caiulo, P. Malcovati, C. Bona and F. Maloberti, ”Novel circuit solutions for rail-to-rail CMOS buffer,” Proceedings of ISCAS’95 - International Symposium on Circuits and Systems, Seattle, WA, USA, 1995, pp. 1980-1983 vol.3, doi: 10.1109/ISCAS.1995.523809.

[48] M. Venkatasubramanian and V. D. Agrawal, ”A New Test Vector Search Algorithm for a Single Stuck-at Fault Using Probabilistic Correlation,” 2014 IEEE 23rd North Atlantic Test Workshop, Johnson City, NY, USA, 2014, pp. 57-60, doi: 10.1109/NATW.2014.20.

[49] T. Dunlap, Z. Lu and G. Qu, "Towards Finding the Sources of Polymorphism in Polymorphic Gates (Invited)," 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC), Incheon, Korea, Republic of, 2024, pp. 319-324.

[50] R. Porter, S. J. Stone, Y. C. Kim, J. T. McDonald and L. A. Starman, "Dynamic Polymorphic Reconfiguration for anti-tamper circuits," 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, 2009, pp. 493-497, doi: 10.1109/FPL.2009.5272469.

[51] R. D. Newbould, J. D. Carothers, J. J. Rodriguez and W. T. Holman, "A hierarchy of physical design watermarking schemes for intellectual property protection of IC designs," 2002 IEEE International Symposium on Circuits and Systems. Proceedings, Phoenix-Scottsdale, AZ, USA, 2002, pp. IV-IV, doi: 10.1109/ISCAS.2002.1010594.

[52] B. Colombier and L. Bossuet, "Survey of hardware protection of design data for integrated circuits and intellectual properties", Computers & Digital Techniques IET, vol. 8, no. 6, pp. 274-287, 2014.

[53] F. Courbon, P. Loubet-Moundi, J. J. A. Fournier and A. Tria, "Increasing the efficiency of laser fault injections using fast gate level reverse engineering," 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), Arlington, VA, USA, 2014, pp. 60-63, doi: 10.1109/HST.2014.6855569.

[54] M. S. Parihar, D. Ghosh and A. Kranti, "Ultra Low Power Junctionless MOSFETs for Subthreshold Logic Applications," in IEEE Transactions on Electron Devices, vol. 60, no. 5, pp. 1540-1546, May 2013, doi: 10.1109/TED.2013.2253324.

[55] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," in Proc. of the International Symposium on Circuits and Systems, 1985, pp. 663-698.

[56] S. Rai, A. Rupani, P. Nath and A. Kumar, "Hardware Watermarking Using Polymorphic Inverter Designs Based On Reconfigurable Nanotechnologies," 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA, 2019, pp. 663-669, doi: 10.1109/ISVLSI.2019.00123.

# List of Publications


[1] T. Dunlap, Z. Lu, G. Qu, "Towards Finding the Sources of Polymorphism in Polymorphic Gates," 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC), Incheon, Korea (South), 2024, pp. 319-324.

[2] T. Dunlap, O. Aramoon, G. Qu, T. Wang, X. Cui, and D. Yu, "A Novel Circuit Authentication Scheme Based on Partial Polymorphic Gates," 2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Shanghai, China, 2021, pp. 1-4.

[3] T. Dunlap, G. Qu and J. Lai, "A Polymorphic Circuit Interoperability Framework," 2019 IEEE 13th International Conference on ASIC (ASICON), Chongqing, China, 2019, pp. 1-4

[4] T. Wang, X. Cui, D. Yu, O. Aramoon, T. Dunlap, G. Qu, X. Cui: A Novel Polymorphic Gate Based Circuit Fingerprinting Technique. ACM Great Lakes Symposium on VLSI 2018: 141-146

[5] T. Wang, X. Cui, D. Yu, O. Aramoon, T. Dunlap, G. Qu, X. Cu, "Polymorphic gate based IC watermarking techniques," 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, Korea (South), 2018, pp. 90-96, doi: 10.1109/ASPDAC.2018.8297288.

[6] Q. Wang, T. Dunlap, Y. Cho and G. Qu, "DoS attacks and countermeasures on network devices," 2017 26th Wireless and Optical Communication Conference (WOCC), Newark, NJ, USA, 2017, pp. 1-6, doi: 10.1109/WOCC.2017.7928974.