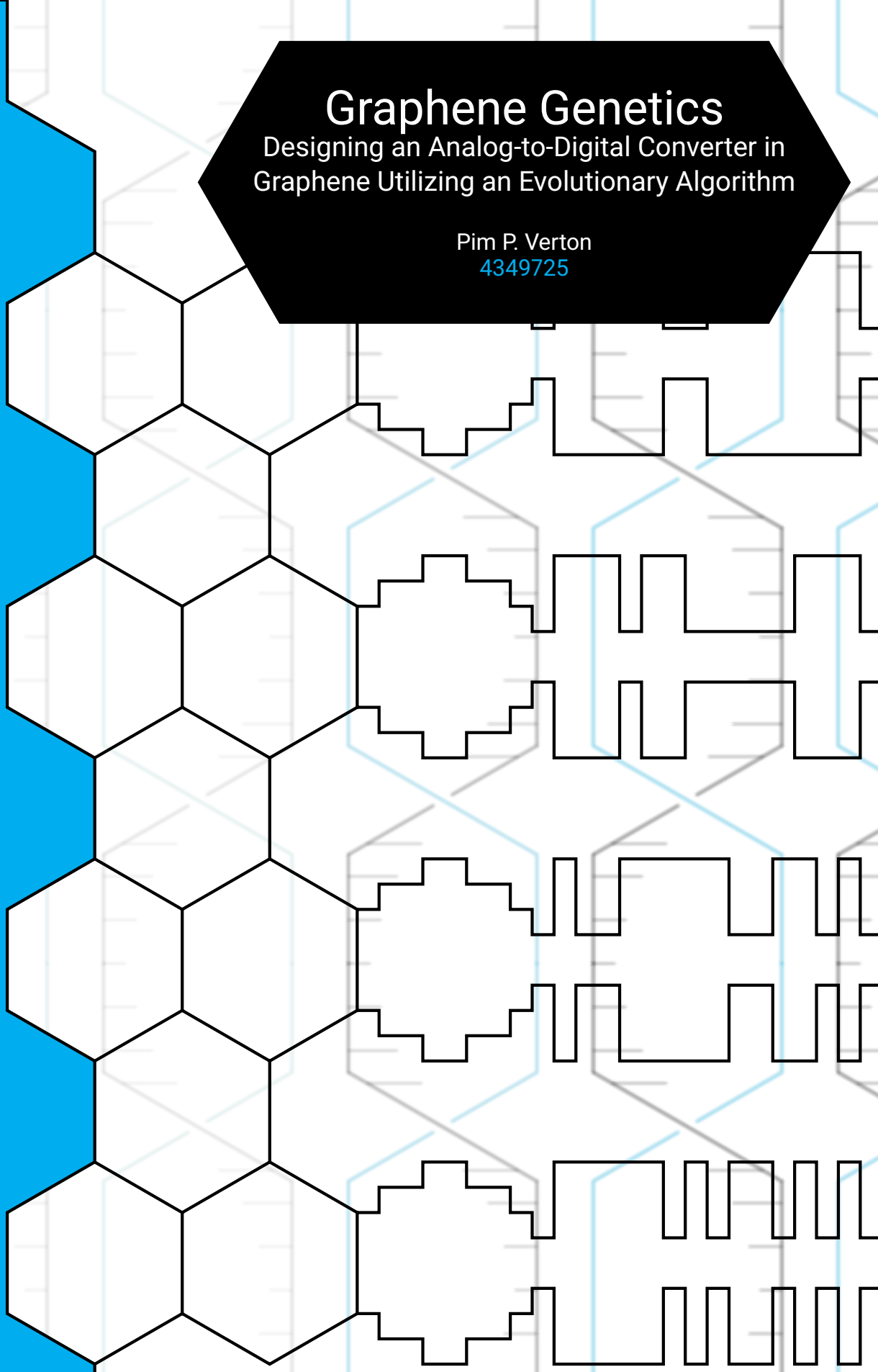
Graphene Genetics

Designing an Analog-to-Digital Converter in Graphene Utilizing an Evolutionary Algorithm

Pim P. Verton
4349725



Master of Science Thesis

TU Delft Delft University of Technology

Challenge the future

Graphene Genetics

Designing an Analog-to-Digital Converter in Graphene Utilizing an Evolutionary Algorithm

by

Pim P. Verton

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Thursday April 25th, 2024 at 13:00.

Student number:	4349725	
Project duration:	September 2020–March 18, 2024	
Thesis committee:	dr. Sorin D. Coțofană,	TU Delft, supervisor
	dr. ir. Sten Vollebregt,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

There's many people I could thank for their continued support in the arduous process that was writing this thesis. But I can only hope for their sake that they never have to read this. Instead I'll dedicate this to my father, who I'm sure won't read this, but I bet he would've liked it.

Abstract

As advances in silicone technology steadily plateau, new avenues of electronics design must be explored. Graphene nanoribbons (GNRs) are a potential solution. Current GNR designs require wasteful exhaustive searches for the required device topologies, limiting the size of the solution space that can be searched. This is fine for simple circuits, but more complex functionality requires a different approach. This work presents a new way of identifying suitable GNR device topologies for any functionality. It also presents an example of this, demonstrating how a circuit using the resultant GNR devices can outperform conventional, more complex circuits.

The proposed methodology uses an evolutionary algorithm to efficiently search a large solution space of possible GNR device topologies. This is done while only having to simulate the behavior of a minuscule fraction of the device topologies in this solution space. The GNR devices found by this evolutionary algorithm are used to implement a 4-bit analog-to-digital converter (ADC), where each bit of the circuit consists of only a couple of GNR devices, in contrast to the many more transistors required for conventional designs.

The resulting ADC circuit performs better than conventional ADC designs in terms of energy cost, conversion delay, and required circuit area by several orders of magnitude.

Contents

Abstract 5

List of Figures 9

List of Tables 11

1 Introduction 1
- 1.1 Research Questions . . . 4
- 1.2 Thesis Contributions . . . 4
- 1.3 Thesis Structure . . . 4

2 Background 7
- 2.1 Graphene . . . 7
- 2.2 Evolutionary Algorithms . . . 16
- 2.3 Analog-to-Digital Converters . . . 20

3 ADC and Algorithm Design 27
- 3.1 The ADC Circuit . . . 27
- 3.2 The Evolutionary Algorithm . . . 32
  - 3.2.1 Creating random GNR devices . . . 36
- 3.3 The Fitness Function . . . 37
  - 3.3.1 Frequency Match . . . 37
  - 3.3.2 Frequency Mismatch . . . 38
  - 3.3.3 Phase Match . . . 40
  - 3.3.4 Amplitude Variance . . . 43
  - 3.3.5 Period Variance . . . 44
  - 3.3.6 Duty Cycle . . . 45
  - 3.3.7 Scale . . . 46
- 3.4 Expectations . . . 47

4 Results 49
- 4.1 The Evolutionary Process . . . 49
- 4.2 Found Devices . . . 55
  - 4.2.1 The pulldown device of bit 1 of the ADC . . . 55
  - 4.2.2 The pulldown device of bit 3 of the ADC . . . 57
- 4.3 The ADC Circuit . . . 59
  - 4.3.1 Bit 1 of the ADC . . . 60
  - 4.3.2 Bit 3 of the ADC . . . 63
  - 4.3.3 Amplification . . . 65
  - 4.3.4 The Full ADC Circuit . . . 68
  - 4.3.5 ADC circuit with amplification . . . 71
  - 4.3.6 Metrics and the state of the art . . . 74
- 4.4 ADCs with Larger Resolution . . . 77

5 Conclusion 81
- 5.1 Research questions . . . 82

5.2 Future Work. 83
Bibliography 85

List of Figures

2.1 The unit cells of the graphene lattice as a bulk material, annotated. 8
2.2 Unit rows and columns in a strip of graphene. 9
2.3 The different GNR shape families defined by Jiang et al. [31]. 10
2.4 Each of the contacts applied to a strip of graphene to create a GNR device. 11
2.5 An example GNR device, schematically. 12
2.6 An example GNR device as a circuit element. 12
2.7 An example conduction map plot 12
2.8 A complementary-pair GNR circuit. 13
2.9 The functional behavior of an example complementary circuit. 15
2.10 An example fitness landscape. 18
2.11 Some initial points on a fitness landscape. 19
2.12 Reproduction on the fitness landscape. 19
2.13 The final state of the evolutionary process on the fitness landscape. 20
2.14 The mapping of analog values to digital symbols performed by an ADC. 21
2.15 An example direct conversion ADC circuit. 22
2.16 The equivalent circuit model for a GNR device. 24

3.1 The required transfer function of each bit of the ADC. 29
3.2 A schematic circuit of the 4-bit ADC. 29
3.3 The required conductance values of the ADC GNR devices. 30
3.4 Relative and absolute conductance thresholds. 31
3.5 A schematic representation of the evolutionary algorithm. 33
3.6 The shapes defined by Jiang et al. [31]. 36
3.7 A GNR device with a very sharp peak frequency. 39
3.8 A GNR device with a scattered frequency spectrum. 39
3.9 Two GNR devices whose phases closely match. 41
3.10 Two GNR devices whose phases don't closely match. 42
3.11 A GNR device with a low relative amplitude variance. 44
3.12 A GNR device with a high absolute amplitude variance. 44
3.13 A GNR device with a low relative period variance. 45
3.14 A GNR device with a high absolute period variance. 45
3.15 A GNR device with a high duty cycle score. 46
3.16 A GNR device with a low duty cycle score. 46
3.17 A GNR device with a high scale score. 47
3.18 A GNR device with a low scale score. 47

4.1 The progression of found device fitness throughout the search. 50
4.2 The self-similarity of devices found through the evolutionary algorithm. 52
4.3 The similarity between the device pool of the final generation of each epoch. 54
4.4 Some candidate devices for the bit 1 PDN device. 56
4.5 Some candidate devices for the bit 3 PDN device. 58
4.6 The complementary GNR device circuit. 60
4.7 The circuit and corresponding SPICE simulation of the bit 1 subcircuit. 61

4.8 The circuit and corresponding SPICE simulation of the bit 3 subcircuit. 64
4.9 The bit 1 subcircuit with an amplification stage. 66
4.10 The bit 1 subcircuit with multiple amplification stages. 67
4.11 A behavioral simulation of the amplification stage. 68
4.12 The full 4-bit ADC circuit. 69
4.13 A SPICE simulation of the ADC's outputs and their digital interpretation. 70
4.14 The ADC circuit with amplification. 72
4.15 The combined ADC output compared to its target output. 73
4.16 The distribution of simulation times for devices of each bit of the ADC circuit. 78
4.17 The distribution of device size for devices of each bit of the ADC circuit. 79

List of Tables

3.1 The desired digital symbol outputs for each analog input value of the ADC. 28

4.1 Noise metrics for the ADC circuit with multiple amplification stages. 74
4.2 A comparison between the ADC's subcircuits and Jiang et al. [32]'s GNR Boolean gates. 75
4.3 Metrics of the GNR ADC as compared to other ADCs. 76
4.4 Error metrics of the ADC circuit with fewer bits of resolution. 77

1

Introduction

The capabilities of electronics have grown exponentially for decades. This was first described by Intel co-founder Gordon Moore[44] in 1965, who noted that the cost of manufacturing a single transistor went down by half approximately every year. Simultaneously, the number of transistors in a typical integrated circuit tended to double in that same timeframe. In other words, typical integrated circuits got twice as complex every year at about the same manufacturing price. This trend was later called Moore's law, and has more or less held ever since.

For a long time, smaller transistors translated directly to faster computation. As noted by Dennard et al., a transistor of smaller size could operate at smaller voltages and currents, and had smaller capacitances, which lead to lower propagation delay and therefore higher switching frequency [13].

In time, however, these spectacular trends have leveled off [3, 24]. As such, new innovations were needed to keep shrinking our transistors and to keep speeding up our electronics. Since then, we've moved from constant field scaling [2, 13, 41] to constant voltage scaling [8] and beyond [3]. We went from conventional planar MOSFETs with the gate on top of the channel, to finFETs [25] with the gate covering multiple sides of the channel, to GAAFETs [37], with the gate all around (GAA) the channel. These innovations were aimed at improving the transistors themselves, independent of concurrent efforts to make the systems consisting of these transistors more intelligently designed to maximize efficiency.

Research both into improving the performance of transistors and into designing ever more efficient computing systems utilizing these transistors complement each other. Together, these two avenues of research have allowed for spectacular growth in the capability of electronics. Advancements in fields like artificial intelligence, big data, and climatology, and many others, are powered by this growth in computational capacity. In many ways, industrial society has even grown to rely on this growth for its future [33].

However, limits to this growth are looming on the horizon. Increasingly, power is becoming a limiting factor [35]: As more is happening in a given chip area, more heat is inevitably produced that needs to be transported away, which is only possible to some extent [55]. On top of this, interconnect is becoming a limiting factor [4], as well as memory access [42, 64], and reliability [50, 66].

One thing we can do to stretch the limits of what we can do with our electronics is to expand away from silicon to new materials. Graphene, an allotrope of molecular carbon, is just one proposed alternative. It has many interesting material properties: it is mechanically and thermally stable [19],

potentially biocompatible [6], and has exceptionally high carrier mobility [5]. Importantly, graphene's condictivity heavily depends on its geometry [56].

We note that graphene fabrication is a whole science in itself, pioneered by Geim and Novoselov [20] in 2007. This work, however, won't delve into that, instead assuming that all graphene devices talked about will be possible to fabricate at some point in the future. Rather than limiting ourselves to the constraints of the manufacturing technology of our time, we will utilize a theoretical graphene behavior model in our investigations.

By designing a graphene field-effect transistor, or GFET, Lemme et al. [38] demonstrated that transistors in graphene are possible, and exhibit better carrier mobility than traditional silicon MOSFETs. As such, any existing silicon design could be made in this new medium, by simply replacing the components but keeping the same circuit design. More fundamentally, though, the existence of graphene transistors demonstrates that the conductance of a graphene channel can be modulated by an externally applied electric field. After all, that is what a transistor essentially does.

Jiang et al. [28] took this concept a step further, designing Boolean complementary logic gates where the pull-up network and pull-down network are implemented by a single GNR device each. These GNR devices consist of a strip of graphene carved into a specific shape, with electrostatic voltages applied by input gates modulating the channel conductance. Note that in a conventional CMOS implementation of these circuits, the functionality of one of these GNR devices would be implemented by several transistors each. In doing this, Jiang et al. [28] demonstrated that graphene devices have the potential to do more complex things than simple transistor switches. Wang et al. [59] and Dumitru et al. [14] then continued on this way of investigation by utilizing GNR devices to implement complex analog behavior: respectively, spiking neurons and digital-to-analog converters. These GNR devices would, in conventional CMOS, be implemented by much larger analog circuits [62, 63].

This previous research suggests that complex Boolean and analog functions can be evaluated by means of a specific pair of complementary GNR devices. This opens up many possibilities for the implementation of complex behavior in simple graphene circuits, the functionality of which would require many more transistors in conventional CMOS. The remaining question then, is how to find the GNR devices that implement a given desired functionality.

Up to date, there isn't any analytical model available to relate the geometry of a given GNR device to its specific behavior. Datta [12] devised a numerical model, based on the Non-Equilibrium Green's Function–Landauer formalism, which evaluates the static current through a specific GNR device at certain conditions such as external biasing. While this model can describe the behavior of a GNR device with a given geometry, one would ideally want to do the reverse. That is, to somehow determine the geometry of a device that can exhibit a given desired behavior. Unfortunately no way of doing this has been devised yet.

In previous works[14, 30, 31, 59], the strategy seems to be one of brute force: A certain solution space of potential device geometries is defined first. The behavior of each device in that space is then simulated, whereupon the best candidate is selected. Since the simulation model is quite computationally expensive, this approach will only get you so far. A good approach would be to somehow search for the required device much more efficiently. One possible, relatively simple approach, is to drive the search for GNR geometries by means of a so-called evolutionary algorithm [67].

An evolutionary algorithm is a class of optimization algorithms inspired by the natural process of evolution [11]. The mechanics of natural evolution are followed to cleverly search for the solution to some problem. Evolutionary algorithms are often a good strategy when not much is known about

the structure of the solution space. In other words, when there is no rigorous understanding of what the best solution is going to look like. In the case of the GNR device, this applies: we have no idea how to predict which device geometries lead to which behavior.

Evolutionary algorithms of different types exist, such as genetic algorithms, evolutionary programming, and evolution strategies [61]. These have been used since the sixties [17, 48, 51] in many different applications [54]. A few examples of these are Lv et al. [40], who used a genetic algorithm to optimize the layout of a solar cell array; Sepahvand et al. [53], who used genetic programming for handwritten character recognition; and Li et al. [39], who used differential evolution to synthesize an optimal design for a dipole antenna array.

In this work, we use an evolutionary algorithm to efficiently find the required GNR device geometry to implement some desired functionality. To demonstrate the capabilities of this evolutionary algorithm, we utilize it to design a four-bit ADC circuit.

An ADC is a circuit that converts some analog input signal into a digital representation of the value of that signal. They are generally used for interactions between the outside world (which is generally analog) and digital computer systems. For example, when an image sensor in a digital camera detects light, it produces an analog electronic signal, which is then converted to digital information to be processed and stored [16, 46]. Similarly, audio data picked up by a microphone is digitized when recording, and antenna systems in telecommunication convert analog signals into the digital information encoded therein. Because of this broad applicability, there's a lot of demand for different types of ADCs.

Specifically, there's demand for ADCs with low cost in terms of power. For example, recent designs of high frequency millimeter wave receivers require large arrays of sensors, each of which produce analog signals which need many ADCs to be digitized [1] within a certain power budget. Recent designs have tackled power limitations by making use of lower resolution ADCs [10, 43, 52]. To stay within reasonable power budgets, tradeoffs are made between having fewer ADCs by combining analog signals, or having lower resolution ADCs which require less power but also capture less information.

In addition to gaining information from the outside world, sometimes the analog domain is more suited for certain operations in a computation system. For example, Haensch et al. [23] proposes a system where a neural network is evaluated by means of an analog circuit to be able to effectively perform all operations at the same time. The output signals from this neural network then need to be converted back to the digital domain by means of ADCs. Hu et al. [26] and Hughes et al. [27] rely on a similar approach. Crucially, Haensch et al. alludes to the benefits to be gained from even further hybridization: some steps in the algorithm are more efficiently done in the analog domain, while other steps are more efficiently calculated digitally, the tradeoff being dominated by the cost of conversion between the domains. It is easy to imagine a world where conversion between the analog and digital domain is cheap enough (however cost may be defined for a certain context) that these decisions are made at a very granular level.

To implement an ADC circuit's behavior in a circuit of GNR devices would require significantly more complex GNR behavior than simple Boolean gates as in [31]. This necessitates the use of a methodology more efficient than brute force to identify the required device geometry. As such, ADCs are a good test case for our proposal to identify GNR geometries by means of an evolutionary algorithm.

Because single GNR devices can exhibit much more complex behavior than CMOS transistors, we expect that a GNR-based ADC circuit will consist of substantially fewer components than a conventional ADC counterpart. This means the circuit area would be significantly reduced, for one.

But since fewer steps occur in the circuit, the latency and power consumption of the circuit would be much lower as well. This could be very useful for contexts such as the one discussed earlier where many low-resolution ADCs are used, as well as situations where frequent switching between the analog and digital domain is desirable.

Another reason why an ADC is appropriate as a target for our algorithm is that it could neatly bridge the gap between existing GNR circuit designs: with Wang et al. [60]'s analog artificial neurons and Dumitru et al. [14]'s Digital-to-Analog Converter on the one hand, and Jiang et al. [31]'s digital Boolean gates on the other hand. A GNR ADC could then further pave the way towards fully-graphene systems.

1.1. Research Questions

The primary goal of this thesis to design an evolutionary algorithm which identifies GNR device geometries given some desired functionality. The research questions relating to this, in order of importance, are:

- **Can an evolutionary algorithm find GNR geometries to implement some arbitrary functionality?**
- **Would such an algorithm be significantly more efficient at searching the solution space than conventional exhaustive methods?**

As a secondary goal and as a way of testing this algorithm, we design an ADC circuit consisting of GNR devices. The research questions involving this ADC circuit are:

- **Can a circuit consisting of only a few GNR devices function as an effective ADC?**
- **Would such an ADC circuit perform significantly better than state-of-the-art circuits in terms of area, energy, and/or time efficiency?**

1.2. Thesis Contributions

Primarily, this work presents an evolutionary algorithm which, based on some arbitrary fitness function associated with some desired circuit functionality, searches for and finds a suitable GNR device geometry and configuration to implement that functionality. This algorithm improves on the state of the art by only simulating a minuscule fraction of the solution space, allowing for the exploration of a massively larger solution space than could be searched with conventional exhaustive methods.

We use this algorithm to find the GNR devices which together implement a 4-bit ADC circuit which outperforms the state of the art of ADC circuits as follows:

- Nine orders of magnitude less active circuit area,
- Four orders of magnitude higher sample frequency,
- Five orders of magnitude lower power consumption,
- A Walden time efficiency three orders of magnitude higher,
- A Walden energy efficiency eight orders of magnitude higher.

1.3. Thesis Structure

This thesis is structured as follows. Immediately following this introduction is Chapter 2, containing additional background. There, we describe graphene, GNR devices, and GNR device circuits. After

that, we discuss evolutionary algorithms: what they're inspired by and how they work. Finally, we go over ADC circuits designs, their principles, some conventional design styles, and the metrics by which they're generally judged.

The chapter after that, Chapter 3, discusses the designs of our system. First, we describe the design of the ADC circuit, which is of a radically different style than conventional ADC circuit designs: we describe how we want it to behave, how the required GNR devices are supposed to behave, and some new insights in how we can evaluate such a novel ADC design. Then we go into how our evolutionary algorithm operates in general, as well as how we will create GNR device geometries to find the right ones according to the fitness function. After that, we can bridge the gap between our requirements for the ADC circuit and the workings of the evolutionary algorithm by describing the specific fitness function used by our evolutionary algorithm so as to find the desired GNR devices for our circuit. Finally, we briefly discuss some of the things we expect of our algorithm.

Chapter 4, then presents the obtained results. First, the behavior of the evolutionary algorithm itself is dealt with. We see that the quality of GNR devices increases over time, and that the results converge to similar GNR device geometries. Second, the found devices are discussed. We see that these devices behave the way we expect and want them to, and talk about the matter of selecting from several good-but-imperfect GNR device candidates. Third, the ADC circuit consisting of these GNR devices is evaluated. We look separately at two different circuits each producing only one bit of the digital output. Doing so, we note that the circuit for the most-significant-bit can also be used as a sort of digital amplifier circuit, improving the signal quality of each other bit circuit. We then look at the full ADC circuit, as well as an amplified version of it. Having done so, we can evaluate the ADC circuit according to some different metrics and compare it to state-of-the-art conventional ADC designs, showing that our design outperforms its competitor by several orders of magnitude. Finally, we discuss how we could extend on our design in order to create a higher-resolution ADC, discussing some of the hurdles to overcome for that and how one might go about doing so.

The thesis ends with Chapter 5, which concludes the work. This starts with a summary of the work. We then circle back to the research questions defined in this chapter and answer them one by one. Finally, we look at what future related work might look like, discussing some ways to improve, expand on, and utilize both our evolutionary algorithm and our ADC circuit.

2

Background

In this chapter we discuss *graphene* and *graphene nanoribbons* (GNRs), and how these can be utilized in electronic circuits. A significant difficulty exists in finding the specific required GNR topology for a given desired circuit functionality. To deal with this problem, this work utilizes an *evolutionary algorithm* to generate the required GNR topology, so we also discuss evolutionary algorithms and how to use them. Finally, we discuss *analog-to-digital converters* (ADCs), how these conventionally operate and some metrics which we will utilize to compare different ADC designs.

In Chapter 1, we briefly mention some complex topics. In this chapter, we examine these topics more closely, including their foundations and some relevant definitions.

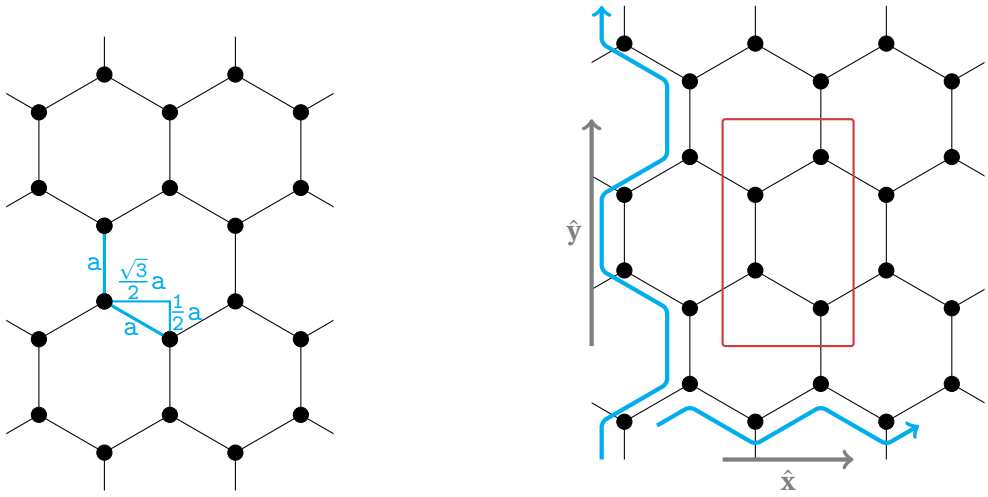
In Section 2.1, we look at *graphene*. First we discuss graphene, *graphene nanoribbons* (GNRs), and the shapes these take. After, we consider GNR devices, what they look like and how they work, and how to characterize their behavior. Finally, we look at GNR device circuits, their topology, how we want them to behave, and what that entails for the GNR devices we need .

Then, in Section 2.2, we explore *evolutionary algorithms*. First we get into biological evolution in nature and *survival of the fittest*. Then we examine how this process is mimicked in evolutionary algorithms for the purpose of optimization, explaining along the way the concepts of *fitness functions* and *fitness landscapes*. Finally we talk briefly about some problems that evolutionary algorithms often encounter.

At the end, in Section 2.3, we explore *analog-to-digital converters* (ADCs), since we are designing an ADC circuit in GNR. We see what an ADC is on a conceptual level and in a practical sense. Then we look at the operation of many conventional ADC circuits, specifically *flash* and *time-domain* (TD) ADC. After, we present which metrics are utilized to judge ADCs by: *resolution* and *signal-to-noise ratio* (SNR), *latency* and *sample rate*, power consumption, *Walden's figures of merit*, and circuit area. We discuss what each of these metrics mean, as well as what typical state of the art designs achieve and how these metrics can be estimated for our design.

2.1. Graphene

In this section, we look at *graphene*. First, we discuss briefly what graphene is as a substance in terms of behavior, dimensionality, and we introduce some definitions. More relevant, we talk specifically about *graphene nanoribbons* (GNRs), what shapes these GNRs can take and how we define the sizes of these shapes. GNR devices make use of these GNRs and we look at those next. This means we look



(a) The relative lengths between carbon atoms are indicated.

(b) The unit cell is indicated in red, and the corresponding unit vectors in x- and y-directions are indicated in gray.

Figure 2.1: Two annotated sections of a bulk *graphene* lattice.

at what these devices look like, how they work, and how we determine their behavior and make use of them. Finally we look at GNR circuits. We discuss how to design a circuit made of GNR devices, how to describe the behavior of such a circuit, and what this means in terms of what we want our GNR devices to look like.

Graphene is a molecular structure consisting of a two-dimensional sheet of carbon atoms in a hexagonal lattice. Its (electrical) properties are very different from other, three-dimensional materials[47]. When talking about graphene as a substance, it is often assumed to be very large relative to this lattice such that the graphene's boundaries and shape can be disregarded. We refer to this unbounded graphene as "bulk" graphene. This substance is very interesting on its own: it is mechanically and thermally stable [19], potentially biocompatible [6], and very strong mechanically [36]. Additionally, its electrical behavior is referred to as "ballistic transport", meaning electrons move through graphene as if completely unimpeded [5].

A section of bulk graphene is presented in Figure 2.1a. The distance between each atom, or the interatomic distance, is indicated as the constant a. These carbon–carbon bonds have a length of around $142\,\mathrm{pm}$ [15]. Since the angle between each bond is $60\,^\circ$, the parallel and perpendicular components of one bond as compared to its neighboring bond have a length of $\frac{1}{2}\,a$ and $\frac{\sqrt{3}}{2}\,a$, respectively. These components are drawn in the same figure.

In this work, a unit cell of bulk graphene is defined as consisting of four atoms, as indicated by the red box, such that the unit cells tessellate the plane orthogonally. Figure 2.1b depicts one unit cell outlined in red. This unit cell has dimensions $\sqrt{3}\,a$ and $3\,a$ in the x- and y-directions, respectively. The unit vectors of these two directions, $\hat{x}$ and $\hat{y}$ are marked in gray in the figure. In the literature, these directions are also referred to as, respectively, the "zigzag" and "armchair" directions, in reference to the shape of the carbon bonds in those directions, as highlighted by cyan arrows in the figure.

In this work, *graphene nanoribbon* (GNR) devices are discussed, which consist of a GNR with certain electrical contacts applied to it in order to create a circuit component. The GNR in question then has a certain specific shape, which greatly affects its behavior. For example, in Jiang et al. [32], the specific shape of otherwise identical GNR device circuits determined the logical function of those circuits.

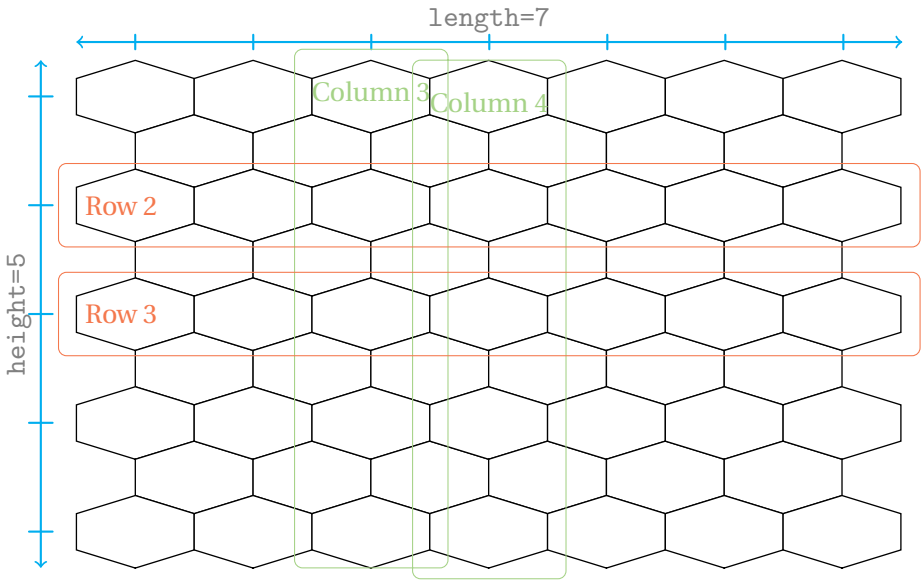


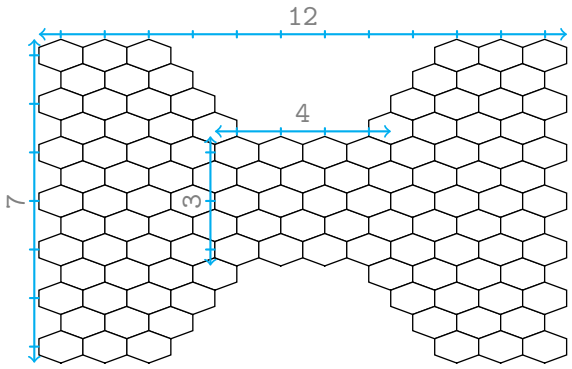
Figure 2.2: A strip of graphene 5 rows high and 7 columns long. The third and fourth row and column are marked in red and green rectangles.

In this work, the size of a graphene shape in the x-direction is referred to as its "length" and its size in the y-direction as its "height". Figure 2.2 displays one of the simplest forms of a GNR: a rectangular strip of graphene. In the x-direction, this strip consists of seven "columns", the third and fourth of which are outlined in green. In the y-direction, the strip consists of five "rows", the second and third of which are outlined in orange. In the rest of this work, we describe the sizes of GNR devices in terms of such rows and columns.

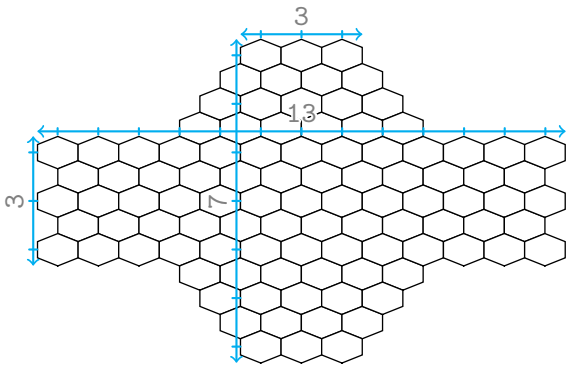
When defining GNR shapes, Jiang et al. [31] defines a few distinct families of shapes, which are presented in Figure 2.3. The "butterfly" shape is defined by a so-called "constriction" area, where some columns of the GNR have a smaller height than the rest, with the surrounding columns sloping down to meet them. Figure 2.3a depicts such a GNR, with a total length of twelve columns, a total height of seven rows, and a constriction of three rows and four columns. Similarly, a "camel" shape is defined by its "bump", which has a larger height than the rest of the GNR, as seen in Figure 2.3b, where the base GNR is three rows by thirteen columns, and there is a bump of seven rows by three columns. Combining these concepts, the "double butterfly" shape of Figure 2.3c has a five-row bump of two columns within a three-row constriction of six columns, in a GNR of in total seven rows and twelve columns.

Having defined a GNR shape, let's talk about how to turn it into a circuit. The GNR functions as a modulated conduction channel. It is suspended between two terminal contacts. In the tradition of transistors, we call these the drain and source contacts, with voltages V_D and V_S. These are ideally modeled as infinitely extending conductors covering a certain number of columns of the GNR over their entire height. Figure 2.4a displays a GNR device with these terminal contacts in covering the outermost two columns on either side. The drain contact is drawn in light blue, and the source contact in darker blue. The terminal contacts of this device are said to have a length of two columns.

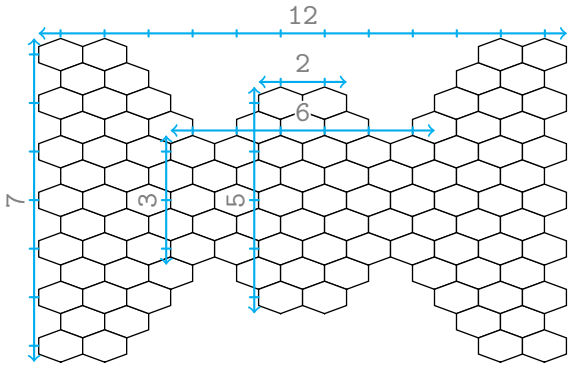
When a voltage difference is applied by these end contacts, a current flows through the GNR, the strength of which is dependant on the GNR shape. This current strength can also be modulated by one or more so-called gate contacts. Similarly to CMOS technology, these gate contacts consist of a conductor which is isolated from the GNR channel by a dielectric layer. The current through the



(a) A GNR with a “butterfly” shape. The length and height of the GNR (12 columns and 7 rows) are indicated, along with the length and height of the constriction area (4 and 3).

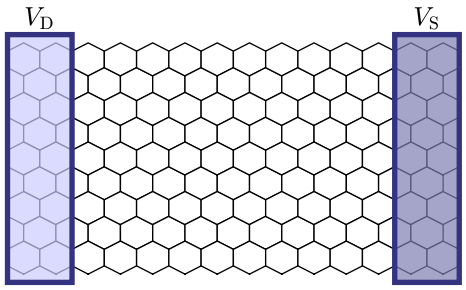


(b) A GNR with a “camel” shape. The GNR length and height (13 and 3) are indicated, as well as those of the bump (3 and 7).

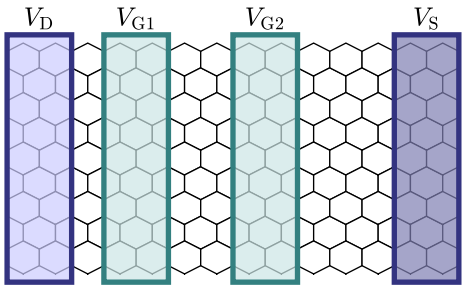


(c) A GNR with a “double butterfly” shape. The length and height of the device (12 and 7) are indicated, as well as those of the constriction (6 and 3) and the bump (2 and 5).

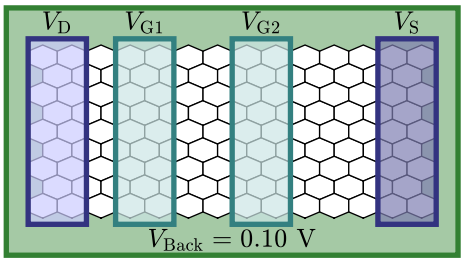
Figure 2.3: The shapes defined by Jiang et al. [31].



(a) The drain and source terminal contacts, with voltages V_D and V_S.



(b) The gate contacts V_{G1} and V_{G2}, modulating the device's current.



(c) The back contact V_{Back}, utilized to apply a bias voltage of 0.10 V.

Figure 2.4: Each of the contacts applied to the GNR.

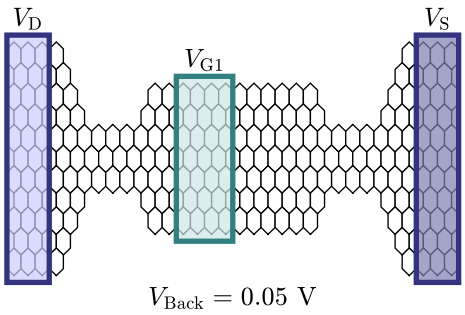
channel then changes depending on the voltage V_G applied to the gate. Figure 2.4b depicts a GNR device with two gate contacts upon which two independent gate voltages, V_{G1} and V_{G2} are applied. The gate contacts are draw in light green.

Finally, a back plate, also isolated from the GNR by a dielectric, can be utilized to apply a bias potential uniformly across the device. This voltage, V_{Back}, also influences the conductive behavior of the GNR device. Figure 2.4c displays this back plate in green, though from here on, this gate voltage is simply noted underneath a device when applicable so it doesn't need to be drawn each time.

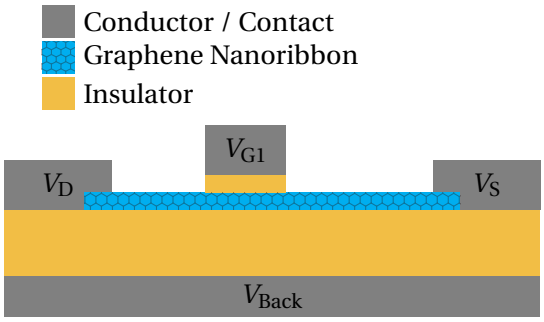
Combining a specifically shaped GNR with these contacts creates a GNR device, which constitutes a circuit element. Figure 2.5 displays such a device schematically, as seen from the top and from the side. In the latter figure, Figure 2.5b, the yellow parts indicate insulating layers separating the contacts which apply a static potential from the GNR. Each contact is colored gray, and the GNR itself is represented in cyan with a hexagonal lattice. Figure 2.6 then presents two different ways of representing this device in a circuit, where the circuit in Figure 2.6a contains this specific device, whereas the symbol in Figure 2.6b can indicate any GNR device with one gate contact.

There is no obvious relationship between the shape of any particular GNR device and its electrical behavior. This makes it quite hard to identify a GNR device able to provide a desired functionality. One way to at least determine the behavior of a GNR device is through an atomistic simulation, as described by Wang et al. [58] and Jiang et al. [29]. This iterative simulation model determines, for a given GNR device with a given V_{Back} and V_G, the conductance between the drain and source terminals.

By performing this simulation repeatedly, a so-called *conduction map* can be generated, mapping different circumstances (in this case device biasing) to the conductance of the GNR device. For

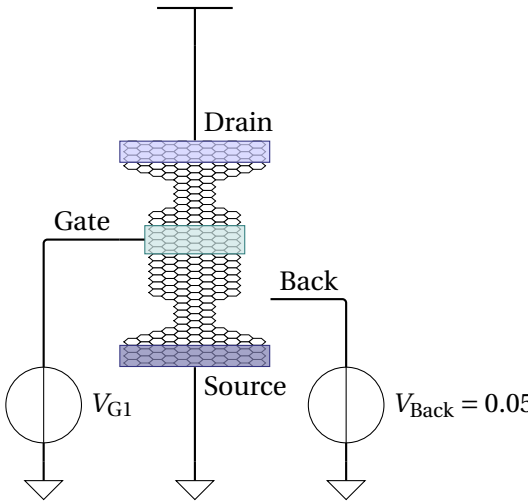


(a) As seen from the top.

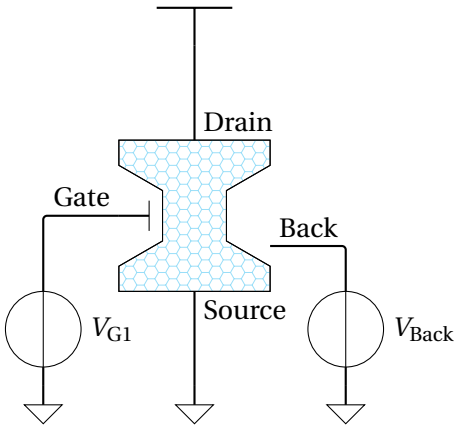


(b) As seen from the side. (Not to scale.)

Figure 2.5: An example GNR device, schematically.



(a) A circuit containing the specific GNR device shown in Figure 2.5.



(b) A circuit containing a GNR device with one gate, symbolically.

Figure 2.6: Example GNR device circuit elements.

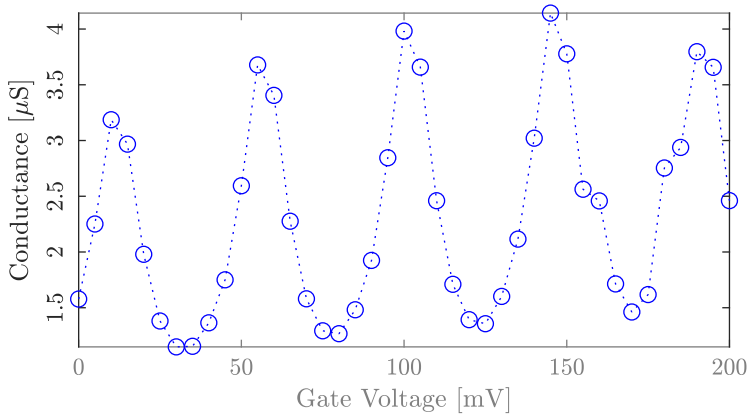


Figure 2.7: An example conduction map, plotted.

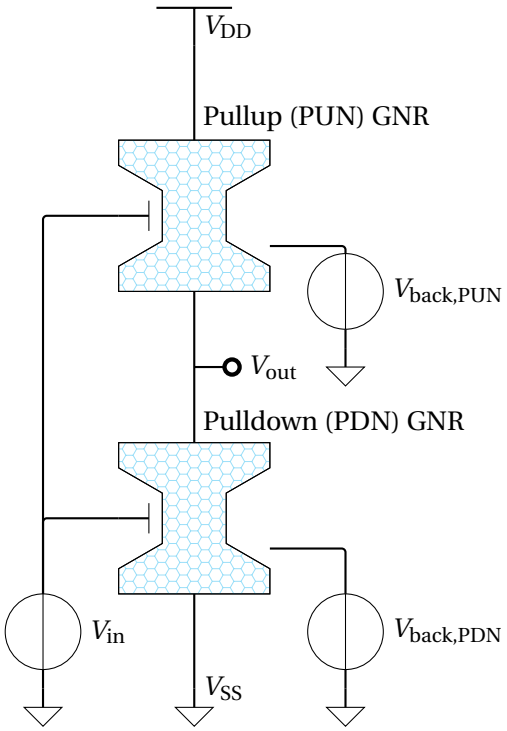


Figure 2.8: The complementary GNR circuit.

example, we can map this conductance for each value of V_G in a certain range with a certain resolution. A *SPICE* simulation can then use this conduction map to simulate the behavior of the device as part of a larger electronic circuit.

Figure 2.7 presents the conduction map of the example device from Figure 2.5. The conductance of this device is plotted with respect to the gate voltage V_G at intervals of 5 mV from 0 mV to 200 mV. A dotted line marks a linear interpolation of these simulated points (indicated as round marks) which is used by the SPICE simulator. In the rest of this work, we generate these conduction maps at higher resolutions and simply plot them as lines.

Given a GNR device, let's look at the circuits we make with them. In a circuit, V_{Back} is assumed to be a constant, while V_G may change, acting like an input signal. The GNR device then acts as a voltage-controlled conductor from drain to source, which is controlled by the gate voltage. This conductance is then called G, which is a function of V_G. Symbolically: $G(V_G)$.

In a complex circuit, it's easiest to work with voltages, rather than conductances. If we can somehow produce a voltage signal containing the behavior of our GNR devices, a subsequent circuit can then use this voltage as an input. This allows for simple interoperation between different subcircuits. So we need to leverage the conductance behavior of the GNR devices into a voltage signal.

One relatively simple way to do this is with a complementary logic circuit, similar to *CMOS* circuits. This requires two GNR devices, which we call the *pull-up network* (PUN) device and the *pull-down network* (PDN) device. These two GNR devices are then configured as in Figure 2.8. The input voltage of the circuit, V_{in}, is connected to the gate terminal of both devices. Supply rails V_{DD} and V_{SS} are connected to the drain terminal of the PUN and the source terminal of the PDN, respectively. In this work, these are always set to 0.2 V and 0.0 V. The output voltage, V_{out}, is then the voltage of the node in between the source terminal of the PUN and the drain terminal of the PDN.

This output voltage then follows Equation (2.1), where G_{PUN}, G_{PDN} are the conductance values of,

respectively, the PUN and PDN. We call this the "*transfer function*". The output can also be expressed as a function of $r(V_{in})$, the ratio between the conductance of the PUN and that of the PDN.

$$\begin{aligned} V_{out}(V_{in}) - V_{SS} &= V_{DD} \cdot \frac{G_{PUN}(V_{in})}{G_{PDN}(V_{in}) + G_{PUN}(V_{in})} \\ &= V_{DD} \cdot \frac{r(V_{in})}{1 + r(V_{in})}, \qquad \text{where } r(V_{in}) = \frac{G_{PUN}(V_{in})}{G_{PDN}(V_{in})} \end{aligned} \tag{2.1}$$

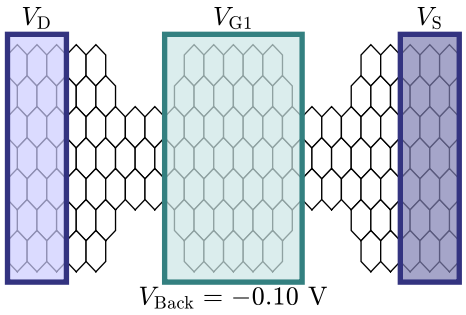
If the value of r is close to zero, the output voltage is close to V_{SS}. As the value of r approaches infinity, the output voltage approached V_{DD}. In this way, the digital values of 0 and 1 can be approached for digital circuits. For those limits of r, the power consumed by this complementary circuit approaches zero since the current running through the circuit approaches zero.

Figure 2.9 presents an example circuit with two GNR devices. Figures 2.9a and 2.9c depict the PUN and PDN devices themselves, whose conduction maps are plotted in Figures 2.9b and 2.9d. Figure 2.9e then displays the transfer function of the circuit. Note that the shapes of both conduction maps are visible in the transfer function: the sharp peaks of the PDN device's conductance are visible as sharp dips in the transfer function, while the generally increasing conductance and the sharp jumps of the PUN device's conductance are noticeable in the transfer function too. Note also that when the PUN's conductance overtakes that of the PDN, the PDN device's features become less prominent.

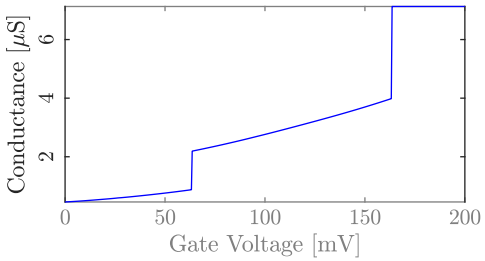
For best results, the conduction maps of the PUN and PDN should always be complementary to each other. That is, for a certain value of V_{in}, if the conductance of the PUN is high, the conductance of the PDN should then be low, and vice versa. If this is not the case, the circuit doesn't work as well. If both conductances are high simultaneously, the circuit behaves like a short circuit and a lot of power is consumed as current flows through both GNR devices directly from drain to source. If both conductances are low, very little current will be able to flow through either side of the circuit. In both cases, little current is able to flow to the output and drive the next stage of the pipeline: the output is "floating".

In all these cases, the terms "high" and "low" depend on the desired output signal amplitude and current. For example, to reach an output voltage of 80 % of V_{DD} or more, r needs to equal or exceed 4, whereas for an output voltage of 99 % of V_{DD}, r needs to be at least 99.

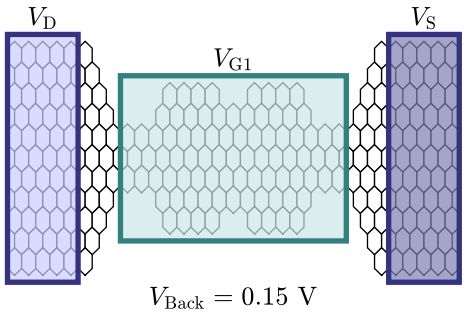
In order to implement some desired functionality (i.e. some desired transfer function), GNR devices with the exact right conduction maps need to be found. In this work, we use an evolutionary algorithm to find these for any arbitrary functionality. In the next section, we discuss what evolutionary algorithms are and how we can use them.



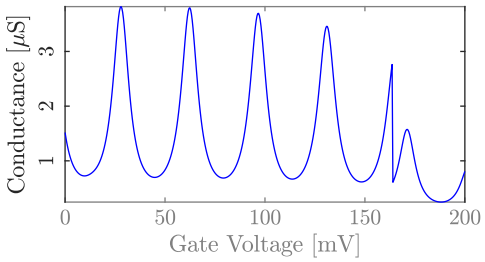
(a) An example PUN GNR device.



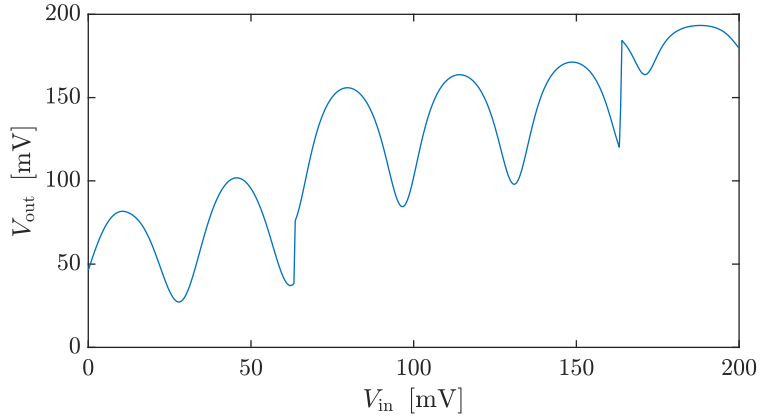
(b) The conduction map of the PUN GNR device.



(c) An example PDN GNR device.



(d) The conduction map of the PDN GNR device.



(e) The transfer function of the circuit combining the the GNR devices.

Figure 2.9: The functional behavior of an example complementary circuit.

2.2. Evolutionary Algorithms

In this section, we discuss *evolutionary algorithms*. To do so, we first explain how evolution works in nature, dealing with *natural selection* and *reproduction*. After that, we explain how evolutionary algorithms try to mimic this process to solve practical optimization problems. We do so first in abstract terms, then with a slightly more concrete example, and finally with a very simplified graphical explanation. We conclude with some remarks on local optima and how evolutionary algorithms may be trapped in them.

In nature, evolution is the process by which organisms slowly, over many generations, take on new forms. A simplified model of natural evolution can be said to rely on the concepts of reproduction, mutation, fitness, and selection. It would operate as follows.

A group of organisms, which we call a *population* live in some natural environment. Every once in a while, these organisms *reproduce*: they create new organisms which are (almost) identical to themselves. In mammals and large organisms, this process happens sexually, meaning two organisms are involved in creating a new one. In this work we ignore this and instead focus on asexual reproduction, where an organism reproduces by itself. It is, after all, an engineering work, and asexual reproduction requires fewer moving parts. The process is not fundamentally affected by this distinction.

As new organisms are created through reproduction and old organisms die, eventually the entire population is replaced by the offspring of the original population. We call this a new *generation*. This new generation consists of a new group of organisms, which are each almost identical to their parents.

We say these new organisms are *almost* identical, since a process called *mutation* occurs. Through processes involving DNA (the specifics of which we won't get into), the child organism is ever so slightly different when compared to its parent. These changes might be beneficial or harmful or inconsequential, but what's important is that they're random.

Over time and many generations, these mutations add up. Given enough time, the population might look wildly different from the original population in various ways. This is especially the case due to a process called *natural selection*. This process influences which traits due to mutation are likely to stay in a population after many generations.

Not every organism reproduces. Some organisms don't find enough food, some become food, many other things can happen. For whichever reason, a subset of organisms do not pass on their genetic material to the next generation, and therefore there are no children who look like them. Natural selection is the process which determines which organisms reproduce and which don't, or which organisms reproduce more than others. In nature, this can consist of food availability, social dynamics, environmental factors, even random chance, as well as a host of other factors.

It would not be quite accurate to say that the organisms which reproduce are "better" than the ones who don't, even if there were any objective measure of how "good" a species is. After all, the selection pressure and selection criteria in a beehive in an apiary are much different than those in a tide pool in the tundra. Instead, we say these organisms are a better *fit* to their circumstances. This is where we get the term *survival of the fittest*, or why we talk about an individual organism's *fitness*.

Even though mutations in each generation are random, there is a trend that appears over time. There is a tendency for organisms which fit the circumstances very well to produce more offspring and organisms which don't fit so well to not reproduce as much. The next generation will then be more similar to the organisms with higher fitness than to the ones with lower fitness. Over many generations, this results in an increasing overall fitness of the organisms in a population. This process

has proven quite successful, which is why scientists and engineers have attempted to emulate it for our own purposes in the form of *evolutionary algorithms*.

In evolutionary algorithms [48], the process of biological evolution is emulated to solve practical problems. In a sense, natural evolution is a process that produces things that conform to a certain set of requirements. The "requirements", in this case, are defined by the circumstances of some natural environment, and organisms are produced over many generations that conform to these requirements. In evolutionary algorithms, some practical problem is formulated as a similar set of requirements, such that solutions to the problem can be produced by the evolutionary algorithm. While there are many variations, we illustrate the fundamental concept of evolutionary algorithms with an elaborate example.

Suppose we have some problem, for which we don't know the solution, nor how to go about solving it. Let's say we're designing a city, and we want to design it in such a way to maximize the happiness of the people in it. It's not as simple as defining some mathematical function which outputs the total happiness of every person in the city and finding the peak of that function. After all, we have no idea what that function would look like or even what the input of that function would be.

What we can do, however, is try something and find out. We can design a city in some way, then simulate its behavior, and see how it would behave. There's a risk that this simulation would be incomplete or incorrect, but we're hoping that it's a good approximation of reality.

If we're using an evolutionary algorithm, here's how we would proceed. First we randomly create a certain number of potential city designs. These are the analogue to biological organisms. This creation should be done in such a way that any conceivable possible design could be randomly created in this way, without any prior assumptions or biases on what makes a "good" design. Of course, where there's concrete limitations, these can be taken into account. We wouldn't want to create cities without houses, or with residential areas at the bottom of the sea, or with no way for anyone to enter or leave, for example.

Given such a design, we can use the simulation mentioned earlier to evaluate how it scores. We can calculate how many people could live in it, how successful they'd be, what their economic production would be or their quality-adjusted life expectancy. We could pick any of these calculated numbers, or combine some or all these numbers somehow into an overall score for the total happiness of the people in the city. This is analogous to the fitness of a biological organism.

Taking this fitness, we can make some selection of potential city designs. For example, we could simply take the cities with the highest fitness scores. These selected cities, we would then let reproduce. Reproduction would consist of copying the city's design, but making some slight incremental changes. For example, we could make a road take a right turn rather than a left turn, or replace a block of houses by a park. What's important is that these changes are incremental and randomizable. These changes are the equivalent of mutations in nature.

For each city in our selection, we can generate some number of "children". This is the population of the next generation. Presumably, the fitness of each of these cities is similar to, but slightly different from their parent cities. After all, a small change like the mentioned mutations won't likely affect the city as a whole very much.

This process then repeats for however many generations are required until some stopping condition is met. This can either be some threshold fitness is reached or some maximum amount of computation time or number of generations have passed.

Like in natural evolution, we can expect a trend to appear of subsequent generations having ever increasing overall fitness. This process likely won't find the absolutely optimal city design, but it likely

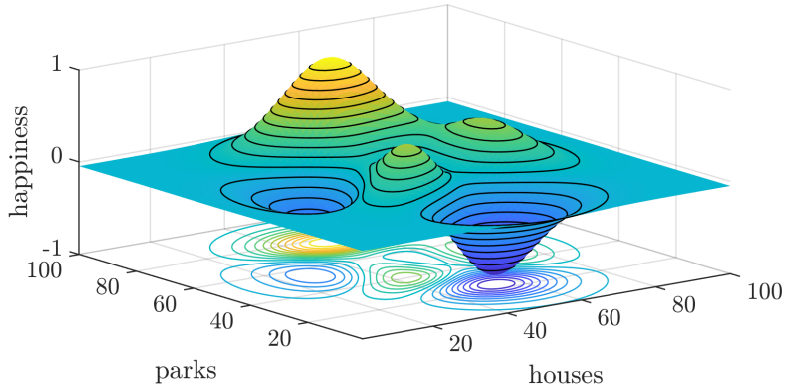


Figure 2.10: The fitness landscape of our example model of a city.

will find a design that's significantly more optimal than the average randomly generated city.

Note that, just like the natural environment of an organism, the way this happiness score—our fitness metric—is calculated, has a large impact on the results produced. After all, if we only looked at economic production, we would see very different resulting cities than if we were only to look at life expectancy, or if we looked at some combination of these factors.

In general, we call the way we calculate our metric of success the *fitness function*. Such a fitness function maps a many-dimensional input (e.g. the design of the city) to a one-dimensional output (e.g. the happiness score) such that each set of inputs maps to exactly one output value. In our example of a city, these dimensions of the input could be any number of things: how many houses are in the city, how many miles of road does it contain, what is built on grid coordinate (x_1, y_1), how many people are in the city council, etcetera.

The fitness function would then contain both the simulation, which produces numbers like population capacity, economic production, life expectancy, etc. and the evaluation of these factors into a total happiness score. Even when the function isn't analytical, we can still call it a function, since it uniquely maps each potential configuration into a single number (i.e. the happiness score or the fitness).

The fitness function is associated with a multidimensional surface. We call this surface the *fitness landscape* or *fitness surface*. To make this easier to talk about, let's reduce the complexity of our example for a bit. Suppose we have a simplified city model with only two inputs: the number of houses built in the city and the number of parks in the city. The fitness function would then somehow map each configuration of these inputs to exactly one happiness score. Figure 2.10 illustrates what the fitness landscape could then look like. Note that the contour lines, which are drawn in black on the surface, are also drawn in the `happiness=-1` plane for clarity. Let's look at our evolutionary algorithm in terms of this fitness landscape. Note that in a real situation, we would not know the shape of this fitness landscape beforehand.

The first thing our evolutionary algorithm does is to create some random cities. Let's say we create a city with 75 houses and 35 parks, as well as a city of 50 houses and 35 parks, and a city of 35 houses and 50 parks. These cities are marked on the fitness landscape in Figure 2.11 as black dots on the surface and the contour plane at `-1`. As a shorthand, we can refer to them by their coordinates: `(75, 35)`, `(50, 35)`, and `(35, 50)`. We can see that the first city, `(75, 35)`, has a quite low positive happiness at

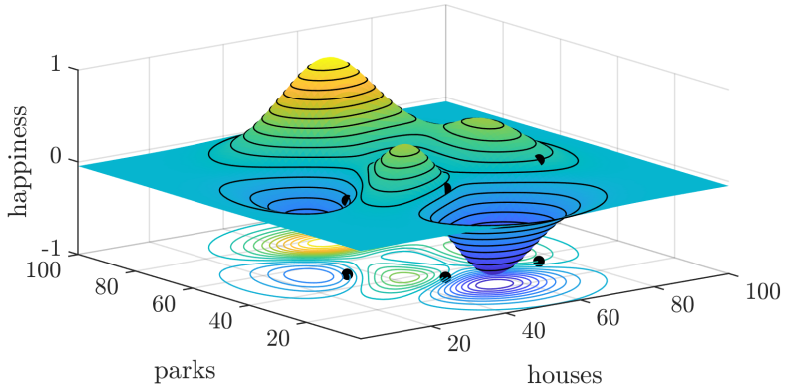


Figure 2.11: A set of hypothetical cities are plotted on the fitness landscape. They are represented as black dots.

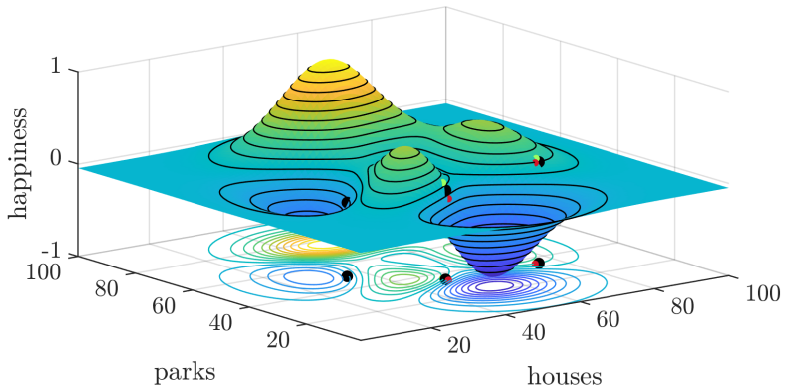


Figure 2.12: Each selected city has produced two mutated children, represented as red or green dots.

0.10, but the other two cities respectively have an even lower happiness of -0.03 and -0.20.

Our algorithm then performs a selection step, selecting only two cities with teh highest happiness score and having them reproduce. The mutations in this case consist of randomly incrementing or decrementing either of the two variables. In terms of the fitness landscape, this corresponds to small steps in either orthogonal direction. We can see the result of this in Figure 2.12, where each city has produced two child cities. In the case of both parents, one of the children got a higher fitness, and the other's fitness decreased with respect to its parent. In other words, one child took a step *up* the hill, and the other took a step *down* the hill. In the figure, the children which increased in fitness are colored in green and the children with decreased fitness are red circles.

This process then keeps going for several generations, each time having the best available candidate cities reproduce to create mutated children, and making a selection from them. Figure 2.13 displays what the end result might look like. The initially selected cities at (75, 35) and (50, 35) are shown as red-outlined black dots. For each generation, the created children are drawn as green or red dots, depending on whether they were selected to produce the subsequent generation. The two top

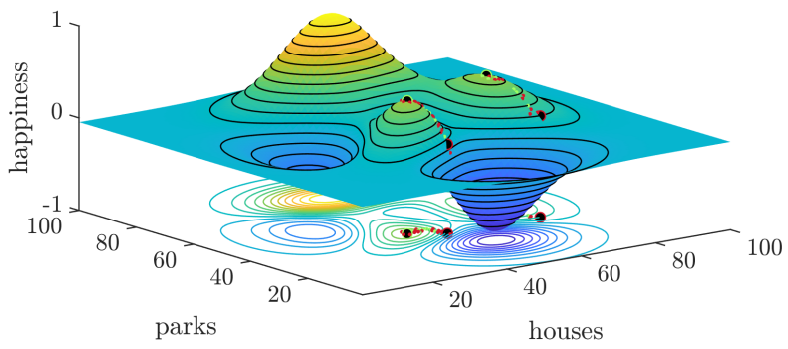


Figure 2.13: The final state of the evolutionary process. The final results found by the algorithm are marked as larger black dots, while each candidate encountered in the search is drawn as a smaller red or green dot, depending on whether it was selected as an improvement.

results are drawn as green-outlined black dots. These final candidates, at (72, 50) and (43, 40), have a happiness score, or fitness, of 0.42 and 0.44, respectively.

In terms of the fitness landscape we can see a sort of meandering path starting at each of these initially created cities and ending up at the top of a hill. The top of this hill is what we call a “local optimum”, which is the best possible candidate within a certain area or locality of the solution space. Often, an evolutionary algorithm has trouble leaving such a local area, since it would require going *down* the hill of the fitness landscape to worse solutions before finding a better solution. In our example, we can see that the algorithm never left these two hills and did not end up at the steeper hill around (50, 77). There it could have found the “global optimum”, the very best candidate within the solution space of possible cities with a happiness score of 1.00.

There are different ways to deal with the problem of finding the global optimum which we don’t get into here. Evolutionary algorithms are quite good at finding a local optimum, however. And in many situations, a good local optimum is perfectly adequate for what one might want. In any case, that’s the hope in this work: To develop an evolutionary algorithm which finds a GNR device geometry which performs some given functionality *well enough*, accepting that it might not be the best possible topology for that functionality.

To be able to talk about this in concrete terms, it’s useful to fill in some of the blanks. So in this work, the "some given functionality" that we try to implement in a GNR circuit is that of an *analog-to-digital converter* (ADC). In the next section, we discuss these ADCs in a bit more detail.

2.3. Analog-to-Digital Converters

In this section, we talk a bit about what an ADC is, what they’re generally used for, and how they are often implemented in rough terms. We also discuss what metrics ADCs are judged by, how state of the art designs score in these metrics, and how we estimate these metrics in this work. First, we handle *resolution* and its related metrics of *signal-to-noise ratio* (SNR) and *effective resolution*. Afterwards comes the cost of an ADC circuit in terms of area. This is followed by the cost of conversion in terms of time—what we call the *propagation delay* or *latency*—and the related metric of *sample rate*. Subsequently, we talk about ADC circuits’ power consumption and *transition energy*. Finally, these metrics are combined into what are called *Walden’s Figures of Merit*.

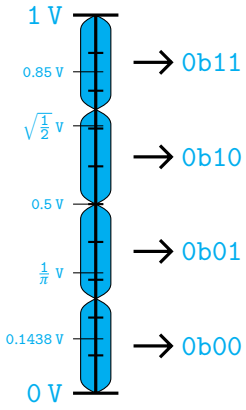


Figure 2.14: An illustration of the mapping of analog values to their corresponding digital symbols performed by an ADC.

An *analog-to-digital converter* (ADC) is, in the most general sense, some system which converts a continuous (i.e. analog) input signal into some discrete (i.e. digital) symbol representing the magnitude of that input signal at some point in time. Commonly, this input signal is some voltage which can take any value within a certain defined range. For example, it could be a signal which can be anywhere between $0\ \mathrm{V}$ and $1\ \mathrm{V}$.

The fact that this signal is "analog" means that there is an uncountably infinite number of potential values this signal can take. Rather than trying to match each possible analog value to a unique digital output symbol, each of a set of output symbols is made to correspond to a certain subset of all input values. For example, in a 2-bit ADC, there are four different output symbols, each of which correspond to a fourth of the input range. All input values between $0\ \mathrm{V}$ and $0.25\ \mathrm{V}$ could correspond to the 2-bit binary symbol 00_2, while values between $0.25\ \mathrm{V}$ and $0.5\ \mathrm{V}$ would then map to 01_2, values between $0.5\ \mathrm{V}$ and $0.75\ \mathrm{V}$ correspond to 10_2, and values between $0.75\ \mathrm{V}$ and $1\ \mathrm{V}$ result in the symbol 11_2 at the output.

This is illustrated in Figure 2.14. A number line from $0\ \mathrm{V}$ to $1\ \mathrm{V}$ is drawn, subdivided into blue regions which respectively map to each of these 2-bit digital symbols. Note that since both $0.1438\ \mathrm{V}$ and $0\ \mathrm{V}$ map to the binary symbol 00_2, these inputs cannot be differentiated by such an ADC. In case more precision is desired, more digital symbols are required.

Conventional ADC designs can be roughly split up into two categories: direct conversion (also called flash), and time-domain conversion. A *direct conversion* ADC operates by means of a series of comparators which directly measure the input signal. Each of them compares the input signal to a different reference voltage, thereby approximately discerning the value of the input signal up to the precision of this set of reference voltages. Figure 2.15 presents an example of such a circuit. A series of resistances generates a set of reference voltages, which get compared to the input signal. The resulting vector of digital comparator results gets compressed to a two-bit number.

Another common family of ADCs are *time-domain* (TD) ADCs. These ADCs operate by converting the input signal to a time-based signal in some way. For example, in *ramp-compare* style TD ADCs, a capacitive node is charged at a steady rate until it matches the input signal. The time taken to charge this node is then the time-domain signal. A time-domain signal can also be for example a *voltage-controlled oscillator*, where the period of oscillation depends on the input signal. With a sufficiently fast clock and digital counting circuitry, the length of this time-domain signal can then be measured to convert the analog input value, through the time domain, to a digital value.

We note that the ADC design implemented in this work is different from either of these design styles.

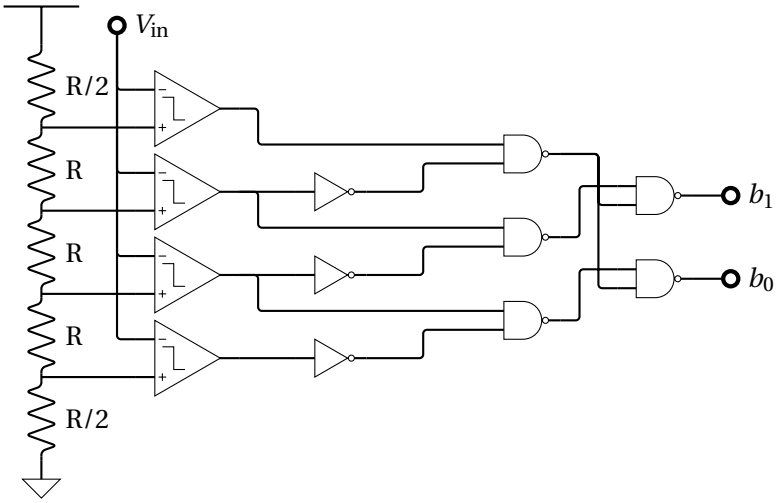


Figure 2.15: An example direct conversion ADC circuit able to discern four different levels of the input signal and convert them to two bits of digital information. Image recreated from Guerber [22].

To be able to make a meaningful comparison, let's talk about the different metrics involved for a bit. Oh [45] compares several different state-of-the-art low-resolution ADC designs in terms of these metrics. We note these scores to give us an idea of the marks we'd like to hit.

The most straightforward metric when considering an ADC implementation is its *resolution*. This is the number of bits of information that an ADC circuit is able to express about the input signal. The compared ADC circuit designs[45] generally have a nominal resolution of six or eight bits. The example ADC in Figure 2.15 has a resolution of two bits. Were it to have one additional bit, it would need twice as many comparators to distinguish twice as many different levels of the input signal, halving the uncertainty.

The error or noise caused by this uncertainty is called *quantization* error or quantization noise. The magnitude of this noise, expressed in decibels, is proportional to the resolution of the ADC in bits.

In an ideal ADC circuit, each segment of the input range can be modeled as a uniform distribution. The input signal can take any value in the segment with equal probability, and the associated output value is the expected average input signal over that range, which is exactly in the center of the segment. For example, the bottom segment in Figure 2.14 can correspond to any value from $0\ \mathrm{V}$ to $0.25\ \mathrm{V}$ with equal likelihood. If the output produced 00_2, you can therefore assume an average input value of $0.125\ \mathrm{V}$. The *root mean square* (RMS) noise (N_{RMS}) in this segment therefore follows the formula in Equation (2.2). The same calculation can be applied to the other segments for the same resulting N_{RMS}.

$$\begin{aligned} N_{\mathrm{RMS}} &= \sqrt{\frac{1}{0.25\ \mathrm{V}} \int_{0\ \mathrm{V}}^{0.25\ \mathrm{V}} (0.125\ \mathrm{V} - v)^2 \,\mathrm{d}v} \\ &= \tfrac{1}{8\sqrt{3}}\ \mathrm{V} \\ &\approx 72\ \mathrm{mV}. \end{aligned} \tag{2.2}$$

To calculate a *signal-to-noise ratio* (SNR) value, we assume the input signal to be uniformly distributed over the input range and shift that input range to have a mean value of zero. This gives an RMS signal

value of $\frac{1}{2\sqrt{3}}$ V, and therefore a SNR of $\frac{1}{4}$.

This generalizes to the formula in Equation (2.3) for the SNR of an n-bit ADC (SNR_n), where A represents the signal amplitude, or half the ADC input range. Expressed in dB, this makes the SNR ($\text{SNR}_{n,\text{dB}}$) equal to $n \cdot 6.02$ dB.

$$\begin{aligned}\text{SNR}_n &= \frac{\text{S}_{\text{RMS}}}{\text{N}_{\text{RMS}}} \\ &= \frac{\sqrt{\frac{1}{2A}\int_{-A}^{A} v^2 \mathrm{d}v}}{\sqrt{\frac{2^n}{2A}\int_{-2^{-n}A}^{2^{-n}A} v^2 \mathrm{d}v}} \\ &= \frac{\frac{A}{\sqrt{3}}}{\frac{A}{2^n \cdot \sqrt{3}}} \\ &= 2^n\end{aligned} \tag{2.3}$$

This is a simplistic model of the precision of a given ADC design, however. A real-world ADC circuit necessarily has more sources of error, such as fabrication process variations or signal noise introduced by other components of the system. By combining all sources of error, a total amount of noise can be determined. This noise can be expressed in terms of an SNR value.

From this SNR noise level, we can calculate what's called the *effective resolution*. This is the resolution of a hypothetical ADC with quantization as its only error source, where this quantization error adds up to the same amount of error. By inverting the formula in Equation (2.3), we can calculate the effective resolution of a given ADC circuit as in Equation (2.4) from its total SNR. Most of the compared ADC circuit designs [45] have an effective resolution between 4 and 5: a few bits lower than their nominal resolution.

$$R_{\text{eff.}} = \frac{\text{SNR}_{\text{dB}}}{6.02} \tag{2.4}$$

In this work, we simulate the behavior of our ADC circuit to determine the noise. Note that the only error sources considered is mismatch between the behavior of the identified GNR devices and their required behavior, and quantization error. Any real-world error sources such as interference or noise aren't modeled.

In addition to conventional metrics of ADC precision like the SNR and effective resolution, we also define some custom metrics that only really makes sense for the specific design in this work. These metrics, the *accuracy* and *ambiguity*, are defined and motivated in Section 3.1, along with the ADC design.

Whereas effective resolution is a metric describing how *precise* an ADC circuit is at determining the value of an analog signal, it's also important to measure the *cost* of this conversion for any practical system. The main measurements of the conversion cost of an ADC circuit are the *power*, *energy*, *time*, and *circuit area*.

The circuit area is a measurement of how much of the surface area of a chip is taken up by that circuit. The lower the area of a given circuit, the more useful circuitry you can put in a given amount of chip space. The ADC circuit designs that Oh [45] compares, have a circuit area on the scale of several hundredths of a square millimeter to a few square millimeters.

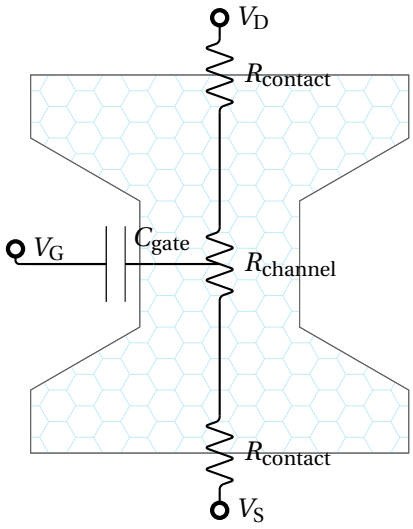


Figure 2.16: The simple equivalent circuit model for a GNR device used here.

The circuit area cost of a hypothetical GNR circuit is hard to judge, since a real design would inevitably involve many unknown factors such as interconnect, clock generation, and the rest of the functionality of a system. As such, it's hard to judge the impact of one ADC circuit on the system's chip circuit area cost. Rather than make a bunch of shaky assumptions, we simply make a very rough lower-bound estimation of what the area cost of a GNR ADC circuit could be. As such, this metric is to be taken with a grain of salt (approx. size $0.1\ \mathrm{mm}^2$).

We calculate this estimation based on the assumption of a rectangular footprint the size of the segment of carved graphene nanoribbon. For example, the device in Figure 2.9a has four rows, for a height of $11\ \mathrm{a}$, or $1.56\ \mathrm{nm}$, and 23 columns, for a length of $23\sqrt{3}\ \mathrm{a}$, or $5.66\ \mathrm{nm}$. From these measurements, we can calculate a very rough estimate of a device footprint of $1.56\ \mathrm{nm} \cdot 5.66\ \mathrm{nm} = 8.84\ \mathrm{nm}^2$. Similarly, we can determine a device footprint of $17\ \mathrm{a} \cdot 32\sqrt{3}\ \mathrm{a} = 19.0\ \mathrm{nm}^2$ for the device in Figure 2.9c. Combining this gives a rough estimation for the complementary circuit of Figure 2.9 of $9.50\ \mathrm{nm}^2 + 19.0\ \mathrm{nm}^2 = 27.8\ \mathrm{nm}^2$.

To be able to make an estimation of the time costs involved in our GNR circuits, we use the same simple circuit model as Jiang et al. [32]. This model relies on a channel resistance R_{channel}, a contact resistance R_{contact}, and a gate capacitance C_{gate}. Figure 2.16 demonstrates the configuration of these circuit elements in relation to the GNR device. The R_{gate} value is calculated based on the size of the gate contact, whereas R_{contact} is derived from the size of the drain and source contacts. The value of R_{channel} is the resistance of the actual GNR channel, as determined by the numerical simulation (i.e. the conduction map). This value is therefore dependant on the specific circumstances of the GNR device such as gate voltage.

We can use this circuit model to calculate the time cost of a conversion in our GNR ADC circuit. We call this time cost the *latency* or the *propagation delay* (τ_p). This metric measures the time it takes before a change in the input signal is reflected by the output.

As in Jiang et al. [32], we calculate τ_p by the formula $\tau_p = (R_{\text{channel}} + 2R_{\text{contact}}) \cdot C_{\text{gate}}$, similar to standard Elmore delay. In this calculation, we take into account the worst case by assuming the maximum channel resistance value of the conduction map. In the case of complementary circuits, we assume the highest value of the propagation delays of the PUN and PDN GNR components. In a multi-stage GNR circuit, we simply add the propagation delay of each stage.

A closely related metric often used for ADC circuits is the *sample rate* or *sample frequency* (F_s). This measures how many times a second the ADC circuit can convert the analog input signal to a digital output symbol. In other words, how many times the input signal can be sampled in a second. In our circuit, this is simply the inverse of the propagation delay: $F_s = \frac{1}{\tau_p}$, but in conventional ADC designs, different types of parallelism can mean that the relation between these metrics is more complex . In the state of the art [45], we see sample rates around two gigasamples per second ($2\ \frac{\mathrm{GS}}{\mathrm{s}}$).

The power consumption (P) of a GNR device follows from the numeric simulation of its behavior. In state-of-the-art [45] ADC circuits, the power consumption is generally on the order of several tens of milliwatts. We estimate the power usage in this work by simply dividing the squared supply voltage by the sum of the channel resistance and the two contact resistances. For a rough worst-case estimation, we take the minimum channel resistance value of the conduction map. In a complementary circuit or a multi-stage circuit, we add the power costs of each GNR component together.

Combining the power cost and the delay of our circuits gives us the *power-delay product* (PDP), or *transition energy*, E_t. The latter name refers to the fact that it's the energy cost of transitioning from one converted value to another. This can be seen as a measure of the energy required to do some fixed amount of useful work (i.e. extract a certain amount of binary information from an analog signal). We calculate this simply as the product of the worst-case propagation delay and the worst-case power usage, assuming that amount of power is consumed for the entire duration of the conversion time.

To be able to better compare various different ADC circuits as a whole, a combined metric was introduced by Walden [57]. These metrics calculate the number of different quantization levels that can be determined by the ADC circuit for a unit cost of either time or energy. Two variants of this combined metric exist: what we will call the *Walden time efficiency* (P_{Walden}), and *Walden energy efficiency* (F_{Walden}):

$$P_{\text{Walden}} = 2^{R_{\text{eff.}}} \cdot F_s \tag{2.5}$$

$$\begin{aligned} F_{\text{Walden}} &= \frac{P_{\text{Walden}}}{P} \\ &= \frac{2^{R_{\text{eff.}}} \cdot F_s}{P} \end{aligned} \tag{2.6}$$

In the state of the art as gathered by Oh [45], ADC circuits score on the order of one trillion conversion steps per second and ten trillion conversion steps per joule ($P_{\text{Walden}} \approx 1\ \frac{\mathrm{Tstep}}{\mathrm{s}}$, $F_{\text{Walden}} \approx 10\ \frac{\mathrm{Tstep}}{\mathrm{J}}$).

Given these different metrics for judging the merits of an ADC, different metrics might be considered depending on the specific intended use case. In this work, we judge our ADC design by these metrics to allow us to see how it performs.

Having discussed the metrics used to judge ADC circuits and their conventional designs, as well as the workings of evolutionary algorithms, and of course GNR circuits and graphene, we are ready to move on to the main body of this work. As such, the next chapter discusses our design of an ADC circuit consisting of GNRs, the geometries of which are identified by making use of evolutionary algorithms.

3

ADC and Algorithm Design

In this chapter we discuss the design, first of our GNR ADC circuit and the functionality required of its constituent GNR devices, and then of the evolutionary algorithm utilized to identify the required geometries of these GNR devices. Separate from the design of the evolutionary algorithm, we discuss the fitness function utilized by the evolutionary algorithm to evaluate the suitability of GNR devices for this ADC circuit. Finally, we discuss some expectations for the results of this algorithm, the GNR devices it produces, and the ADC circuit they make up together.

Having defined what we're talking about, we're now ready to actually talk about the design of our GNR ADC circuit and evolutionary algorithm. In this chapter we're going to do that.

First off, in Section 3.1, we look at the design of the ADC circuit that we're creating. We see how we want the circuit to behave, what that means for the individual GNR devices we use, and we define some metrics to judge the performance of the subcircuits governing the behavior of individual ADC bits.

Next, in Section 3.2, we discuss the design of the evolutionary algorithm which will identify the required GNR device geometries for the ADC circuit. We start by looking at the global structure of the evolutionary algorithm in terms of functionality. We then discuss the functionality needed for the evolutionary algorithm to specifically look for GNR devices. In particular, this means that we discuss how to create and mutate GNR devices.

The final component of the algorithm, which connects the specifications of the desired ADC circuit GNR devices to the evolutionary algorithm is the fitness function, which gets its own section in Section 3.3. There, we discuss each individual component of the fitness function and how they relate to the GNR device functionality we require.

After all that, there's a small section, Section 3.4, which deals with the expectations we have of the results of the evolutionary algorithm, the ADC circuit, and the behavior of the GNR devices it consists of.

3.1. The ADC Circuit

To be able to talk about the design of our ADC circuit, let's first look at what we want its output to be for each input value. We're designing a 4-bit ADC circuit with an input signal assuming any value from $0.0\ \mathrm{V}$ to $0.2\ \mathrm{V}$. The desired output for each input in this range is displayed in Table 3.1, where a four-bit digital symbol is assigned to each portion of the input signal's domain. Note that the range of the input signal is subdivided into sixteen subranges, each of which is mapped to a unique four-bit

Input signal [mV]		Output symbol		Input signal [mV]		Output symbol	
from	until	binary	hexadecimal	from	until	binary	hexadecimal
0.0	12.5	0000	0	100.0	112.5	1000	8
12.5	25.0	0001	1	112.5	125.0	1001	9
25.0	37.5	0010	2	125.0	137.5	1010	A
37.5	50.0	0011	3	137.5	150.0	1011	B
50.0	62.5	0100	4	150.0	162.5	1100	C
62.5	75.0	0101	5	162.5	175.0	1101	D
75.0	87.5	0110	6	175.0	187.5	1110	E
87.5	100.0	0111	7	187.5	200.0	1111	F

Table 3.1: The mapping of analog input signals of the ADC circuit to their corresponding four-bit digital output symbol.

digital symbol. These output symbols are denoted both in binary and hexadecimal, while the input signal subranges are defined by their boundaries.

When designing a complementary GNR circuit, we consider the transfer function, mentioned previously. This is the mapping between the input voltage signal and the (digital) output signal. As such, to design the ADC circuit we need to design four complementary GNR subcircuits, each of them providing the behavior of one ADC output bit. Therefore, each of these subcircuits has its own transfer function which corresponds to the mapping between the values of the input signal and those of one respective output bit. Before going any further, let's look at these functions for a second.

Figure 3.1 presents four plots stacked on top of each other. Each of these plots represents the function mapping the input signal of the ADC (V_{in}) to the digital value of one bit of the output symbol (b_n, $\forall n \in \{0,1,2,3\}$). We notice immediately the periodicity of these functions: each bit periodically alternates between a low and high value with a varying period length. Since each bit assumes 2^{n+1} different values, the period of repetition for each bit's transfer function works out to be $25 \cdot 2^n$ mV. Within this period, each bit's transfer function behaves much in the same way: the binary output value should be 0 for the first half of the period, and then 1 for the second half. The transitions between each value, in the idealized transfer function, are immediate and discontinuous. The output signal is digital, and therefore discrete, after all.

So the ADC circuit should consist of four separate subcircuits, each converting the analog input signal into one bit of the four-bit output symbol. Each subcircuit implements a transfer function of an output signal which oscillates between a logical low and high value in response to an increasing input signal. The period of this oscillation depends on which bit the subcircuit implements. This circuit is depicted in Figure 3.2. Each subcircuit consists of a complementary pair of GNR devices, so let's see what the conduction maps of these devices should look like.

Whenever we want the binary output of a complementary circuit to be 1, the PUN should have a *high* conductance value and the PDN device should have a *low* conductance value. For a binary 0, the opposite should be true: low conductance for the PUN and high conductance for the PDN.

Figure 3.3 displays the required behavior of the GNR devices. For each PUN (Figure 3.3a) and PDN (Figure 3.3b) GNR device, the desired conductance value G is plotted as a function of the input signal V_{in}. These functions, the conduction maps, capture the relationship between the desired binary output value and the corresponding conductance values for each input value. The vertical axes of these plots aren't labeled, since for now all we know is that we want the conductance value to either be "high" or "low", without specifying any values.

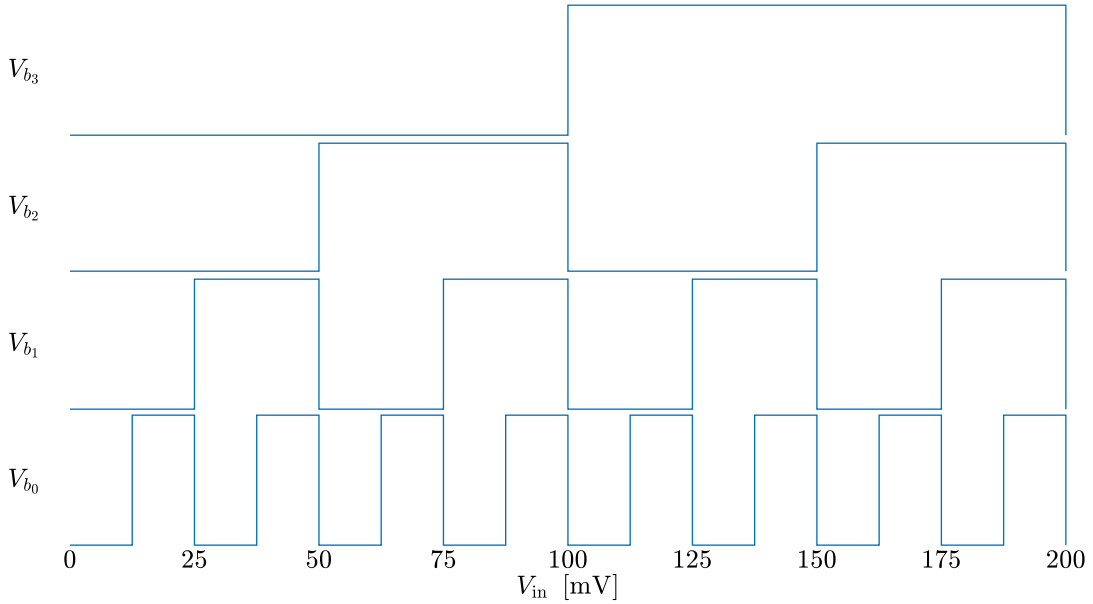


Figure 3.1: The required binary output value of each bit of the ADC as a function of the input signal V_{in}.

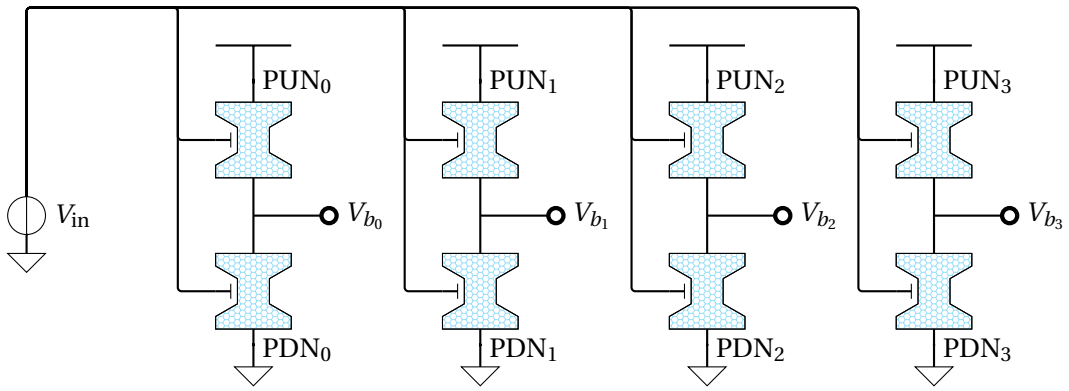
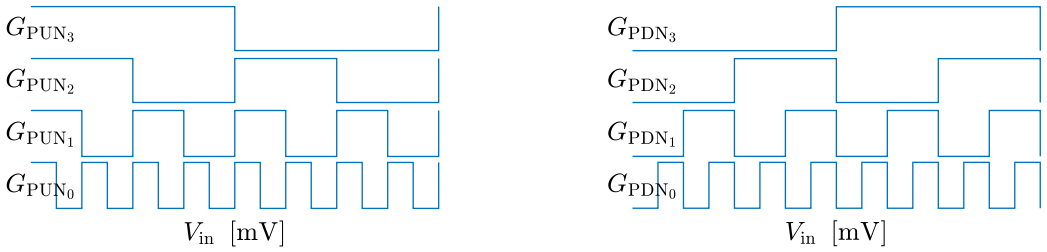


Figure 3.2: The ADC circuit consisting of four complementary pairs of GNR devices, each of which implement the transfer function from the analog input signal to one of the bits of the output symbol.



(a) The required conductance values of each PUN GNR device.

(b) The required conductance values of each PDN GNR device.

Figure 3.3: The values for the conductance value that each GNR device in the ADC should have, as a function of the input voltage V_{in}.

In addition to the value of the conductance of these devices, it's important that these conduction maps actually match each other. That is, for any given input value, *both* conductance values need to be correct: one high and one low. If both conductance values are high, a short circuit is created, whereas two low values won't be able to drive the output to a distinct value. Because of this, it's important to make sure the *transition points* of the devices match up: the points where the conduction map transitions between a low and a high conductance value. These transition points should be the same value in both devices. In the case of, for example, the second-most significant bit ($G_{\mathrm{PUN},2}$ and $G_{\mathrm{PDN},2}$), these transition points should be as close as possible to 50 mV, 100 mV, and 150 mV.

It might actually be important to define the terms of "high" and "low" conductance, so let's talk about this for a bit. We actually define these terms in a few different ways, depending on the context.

First, we can look at a conduction map on its own and use that alone to determine low and high values. We do this by taking the conductance values 40 % and 60 % of the way between the lowest and highest conductance values. These two values form two "relative" thresholds that the conduction map values can be compared to, such that we call the conductance "high" if it's above the upper threshold and "low" if it's below the lower threshold. Conductance values in between the two thresholds we call "ambiguous" or "intermediate". In this way, something can be said about the shape of the conduction map on its own, independent of anything else.

However, the conductances of the PUN and PDN devices in comparison to each other are important. The problem is that at the time when we're finding the right device to act as the PDN, we don't yet know what the PUN device will look like, and vice versa. Because of this, we can't know how the two will behave together in advance. As a way around this problem, we can define high and low conductances in absolute terms, so both devices are judged in the same way.

From a large dataset of simulated GNR devices at different voltage conditions, we can determine a spread of feasible conductance values. This dataset consits of roughly 250 thousand devices, the conduction map of each is simulated at a resolution of at least a hundred datapoints. From this dataset, we can then determine high and low conductance thresholds from *quantile* values such that 40 % of *all* conductance values are below the low threshold and 40 % of conductance values are above the high threshold. We can then use these absolute thresholds to judge the conduction maps of GNR devices relative to those of all other GNR devices. As it turns out, these threshold values are about 0.74 µS and 2.8 µS.

In Figure 3.4, the conduction map of some GNR device is depicted with both of these sets of thresholds indicated. In light red are two dashed lines indicating the relative thresholds at forty and sixty percent of the range of the conduction map. In the same color, there are some triangles pointing up from the

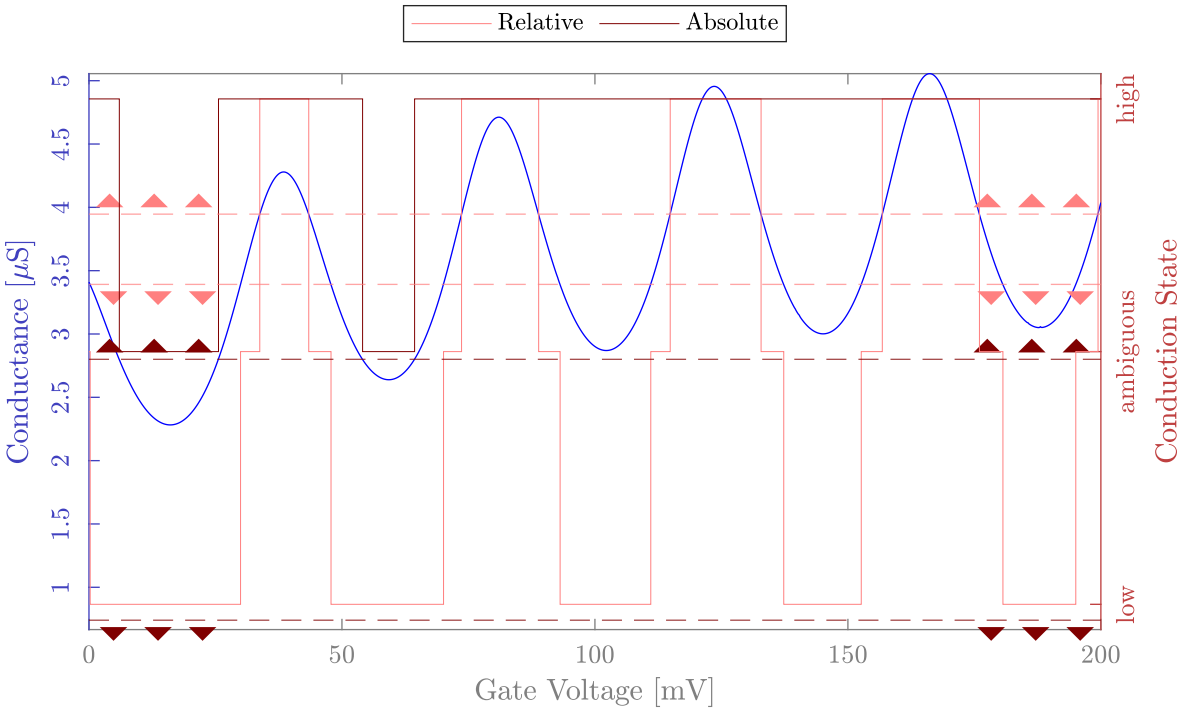


Figure 3.4: In this conduction map, we can see both the relative thresholds and the absolute thresholds, which are used to distinguish low and high conductance values.

upper threshold and down from the lower threshold, to be able to distinguish the two. By comparing the conduction map signal to these thresholds, we can construct a discrete signal which only assumes the values “high”, “low”, and “ambiguous”. This discrete signal is also plotted in the same color as the thresholds.

In the same figure, the absolute thresholds mentioned earlier, at 0.74 µS and 2.8 µS, are also drawn. These are drawn in darker red, including the resulting discretized signal.

Similar to defining high and low conductance values, we can say something about the resulting voltage signal of the complementary GNR circuit. The analog value of this signal is interpreted in some way by any subsequent system to mean one of the two binary logic values. The way this is done is highly dependant on what this subsequent system might look like, but what’s important is that it shouldn’t be ambiguous. If the output voltage signal is exactly halfway between V_{SS} and V_{DD}, who’s to say whether it is supposed to be a 0 or 1, logically. In this work, we’ll assume that values below 25 % of the rail-to-rail voltage are unambiguously interpreted as a logical 0, while values above 75 % of this rail-to-rail voltage are interpreted as a 1. As it turns out, the values we just determined for the absolute conductance thresholds—when plugged into Equation (2.1), as the PDN and PUN conductance values and vice versa—result in just about these voltages, at 0.21 V_{DD} and 0.79 V_{DD}.

Since our design of an ADC is different from many conventional designs in that it implements each bit of the output symbol independently, it’d be nice to be able to judge the merits of each bit on its own. Suppose we find some GNR devices that implement the functionality required for one of the bits’ subcircuits, but they’re not perfect. We can look at the transfer function of the subcircuit to characterize how close this subcircuit is to the ideal. We define two metrics here, as hinted at before in Section 2.3: the *accuracy* and *ambiguity* of each bit’s circuit.

Both of these metrics look at the transfer function of a complementary GNR circuit. Specifically, they look at the binary value of the output signal. To determine this value, the analog output voltage is compared to the output voltage thresholds we just defined of 25 % and 75 % of the rail-to-rail voltage. If the output voltage is below the lower threshold or above the higher upper threshold, the binary value is determined to be either 0 or 1. If, however, the output voltage is between the two thresholds, we call the binary value "ambiguous".

To determine the accuracy of a bit's subcircuit, we compare this binary output value to the required binary value as indicated in Table 3.1. If we do this for every possible input value in the range, we can calculate the percentage of input values that result in the correct output value, as well as the percentage of input values that result in an incorrect or ambiguous output value. We define the accuracy metric of one such bit subcircuit to be the percentage of input values that result in a correct binary output. Similarly, we define the ambiguity metric of a bit subcircuit to be the percentage of input values that result in an ambiguous output value between the voltage thresholds.

Given these metrics for a single bit, we can also combine them into similar metrics for the entire ADC circuit. We can simply take the average accuracy and average ambiguity by taking the average of these individual metrics across each bit's subcircuit.

In addition to that, we can calculate the percentage of input values that result in a correct output value of *all* bits. After all, each bit needs to be correct for the output symbol to be correct. Note that this is a distinct metric due to how the incorrectness of different bits may or may not overlap. We call this the "total" accuracy of the ADC circuit.

The case for ambiguity is similar to that of accuracy, but slightly different: the output symbol is ambiguous if *any* of the constituent bits are ambiguous, since that will render the unambiguous bits meaningless. So to determine the "total" ambiguity of the ADC circuit, the percentage of input values is calculated that result in one or more of the output bits having an ambiguous value.

Having defined the desired behavior of our GNR devices, let's start looking at the evolutionary algorithm that we're using to actually find and identify these devices.

3.2. The Evolutionary Algorithm

To automatically find a set of GNR devices that exhibit the behavior we are looking for, we're using an evolutionary algorithm. This algorithm is going to follow the principles of evolution to find the GNRs we want. In this section we describe this algorithm.

First we discuss the workings of the algorithm in general terms without worrying about the specific implementation of anything. We then discuss these matters of implementation one by one: We start by discussing the way in which we create random devices. Then we discuss how these devices can be mutated to be slightly different each generation. Finally, we look at the details of the fitness function we're using, which determines what makes a GNR device suitable for our purposes.

The rough workings of our algorithm are presented in Figure 3.5, which we'll walk through step by step. The algorithm starts at the black dot at the top and follows the black arrows. Each step of the algorithm is indicated by a rounded gray rectangle. Boxes are drawn around certain subsections of the algorithm to indicate functional units that are repeated multiple times. The "device" box is repeated several times overlaid on each other, indicating that that box is executed many times in parallel. A gray diamond shape indicates a decision which is made, which determines whether to exit the loop or to keep going. On the left of the diagram are gray boxes indicating input parameters. No detailed exploration of the values of these parameters is performed, so we won't discuss what their values should be, other than mentioning the values used in this work.

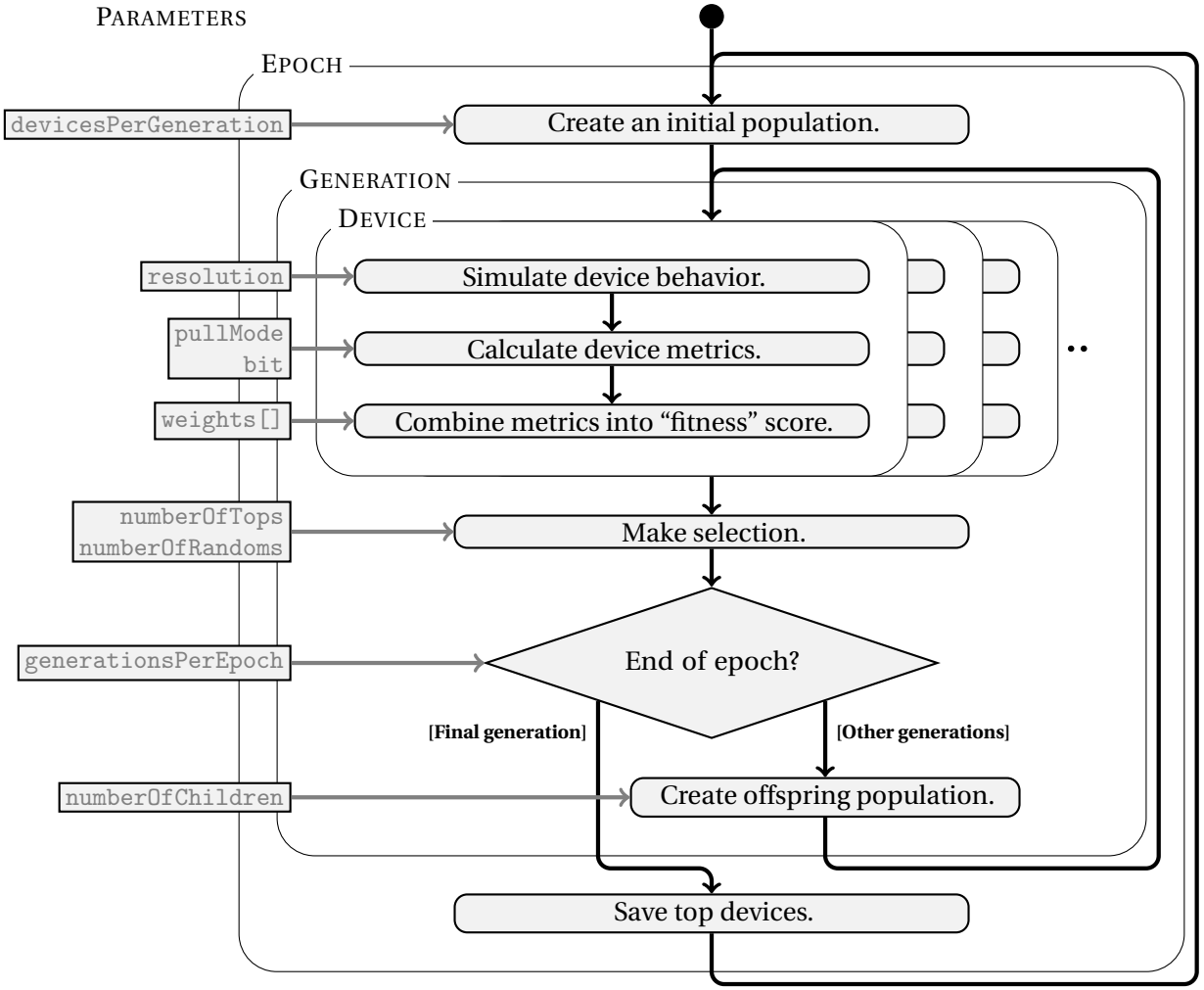


Figure 3.5: A schematic representation of the evolutionary algorithm as we use it.

The first thing we do is to create some amount of random GNR devices. This is the initial device pool, analogous to a population of biological organisms. We'll discuss how to create these random devices later in this section. The parameter `devicesPerGeneration` determines how many devices are in this initial pool, which forms the first generation. In our case, `devicesPerGeneration` = 120.

Having created this pool of devices, each individual device needs to be evaluated. This is highly parallelizable, since each device is completely independent of each other. As a result, many devices can be evaluated at the same time.

The evaluation of each individual device consists of three steps. First, the behavior of the device is determined through an atomistic simulation as described in Section 2.1. This simulation produces a conduction map which maps the voltage V_G at the gate contact of the GNR device to its conductance between the drain and source terminals. This conduction map consists of a certain number of different points, corresponding to as many different values for V_G between 0 V and 200 mV, inclusive. The number of points simulated for this conduction map is determined by the parameter `resolution`. In our case, `resolution` = 100. The devices which are identified as potential good candidates are simulated again later at a higher resolution afterwards to allow for more detailed circuit simulation.

Secondly, the device is evaluated according to the fitness function, which we'll discuss later in this section. This fitness function is different depending on the specific GNR device in the ADC circuit that we're looking for at any given moment. As such, it depends on the parameters `pullMode` and `bit`, which respectively signify whether we're searching for a PDN or PUN device, and which bit of the ADC the device is meant to implement. The evolutionary algorithm runs several times independently with different values for these parameters to search for each required device to construct the full ADC circuit. In any case, the fitness function produces a set of metrics for the GNR device which describe how well the device conforms to a number of independent requirements.

After having calculated the metrics which describe how well the device conforms to these requirements, these metrics need to be combined into a single score, which we'll call the overall device fitness. This fitness score is calculated from the individual metrics by taking a weighted arithmetic average: Each metric is assigned a certain weight, by which it is multiplied. The resulting weighted metric products are all added together to the final score, which is normalized by dividing by the sum of all weights. See Equation (3.1), where each metric M_n for some set of values for n has an associated weight W_n. The values of these weights, determined by the parameter array `weights[]`, determine how important each individual metric is to the overall fitness function. This fitness score then indicates how well the device conforms to all of the requirements at once, or how well it performs the function we want it to.

$$\text{score} = \frac{\sum_{\forall n} M_n \cdot W_n}{\sum_{\forall n} W_n} \tag{3.1}$$

After the fitness score of each device in the pool is determined, a selection is made. According to the parameter `numberOfTops` a certain number of the top scoring devices are selected. In addition to this, a certain number (`numberOfRandoms`) of other devices are randomly selected. These selected devices will be used to create the population of the next generation. The reason the top scoring devices are selected is simply according to the principle of survival of the fittest: if the "best" devices of this generation populate the next generation, over time the population will grow better and better.

The random devices are to allow for a little breadth in the exploration. If we only ever select for the very best device in your pool, we run the risk of getting stuck in a very local optimum. Suppose the fitness landscape is quite bumpy on a small scale. If we just follow the slope of the landscape

by always selecting the top candidates, we will quickly reach the peak of such a bump and never leave it. This happens even if the next bump over might be higher, or if all the bumps are located on the shallow slope of a large mountain. Because of this, it's important to sometimes check the neighborhood of your bump to see if there's anything promising there. This is done in our case by always taking into account a few devices from the population which didn't necessarily score very high, on the chance that their children happen to do better than the current top scorers. In our case, `numberOfTops` = 10 and `numberOfRandoms` = 2 .

Having made this selection, we can proceed one of two ways. Most of the time, this selected subpopulation is used to create a new generation's population. This is done by creating several mutated offspring of each selected device according to the parameter `numberOfChildren`. This parameter is chosen so that

$$(\texttt{numberOfTops} + \texttt{numberOfRandoms}) \cdot \texttt{numberOfChildren} = \texttt{devicesPerGeneration}$$

so each generation's population is the same size. In this case that means `numberOfChildren` = 10.

Later in this section, we discuss how we apply these mutations to the selected devices, but what's important is that each child is similar to, but slightly different from its parent. In this way, each generation's population contains devices which are slightly better and slightly worse than the selected devices from the previous generation, so we can then select the slightly better ones for the next generation.

After a certain number of generations, however, we're likely to have reached a point where new devices won't be better than their parents anymore, since we've reached a local optimum. At this point we can do one of three things: First, we can stop the algorithm, accepting the level of fitness reached by these locally optimal devices. Alternatively, the second option would be to keep going as we are until the random devices selected each generation produce something which is far enough away from the local optimum that it points us to a new, peak in the fitness landscape. In a sense, this is what we do to some extent every generation. After all, that's exactly why we select these random devices. But there's also a third option, which is to start all over again.

In nature, sometimes a mass extinction event happens. Oxygen enters the atmosphere, a meteor hits the planet, industrial society wrecks everything for a little while; any number of things can cause this. The effect is for a large fraction of all species to go extinct in a short amount of time. After such an event, whichever species survived have less competition and are free to thrive, developing new species. In terms of fitness, these surviving species represent a random new location in the fitness landscape from which the process of evolution can start anew. We say it's a *random* location since the fitness function prior to the mass extinction won't have included such an anomalous event. After some time, new local optima will have been reached, which are unrelated to whichever species might have existed prior to the extinction event.

In our algorithm, we can do something similar. By replacing the whole population and creating a random new one, we move to a random new place in the fitness landscape, from where we can find a new local optimum. The expectation is then that this new local optimum is distinct from what we've found before, and hopefully of a higher fitness. We do this every so often, after some number of generations has passed, according to the parameter `generationsPerEpoch`. Misusing a term from geology, we call the time between such "extinction events" an *epoch*. The length of an epoch determines whether the evolutionary process will have had the time to reach a local optimum, as well as how much time is wasted mucking around that local optimum. In our case, we've set it so `generationsPerEpoch` = 15.

At the end of each epoch, the devices which were selected for their fitness are saved. After several epochs, all these devices can be compared again to find the very best ones. In our case, we let the algorithm run for 15 epochs. This allows for different local optima to be reached in different epochs, where the best local optimum would hopefully approach the global optimum. At the end of the chapter, we will get into our expectations a little bit more.

Given the values for each of these parameters, the total amount of GNR devices checked by this algorithm amounts to 27 000 different devices, as seen in Equation (3.2).

$$\texttt{devicesPerGeneration} \cdot \texttt{generationsPerEpoch} \cdot \texttt{numberOfEpochs} = 120 \cdot 15 \cdot 15 = 27\,000 \quad (3.2)$$

Having described the working of the evolutionary algorithm without going into specifics, let's start filling in these blanks. First of all, let's look at GNR devices and how to create them randomly.

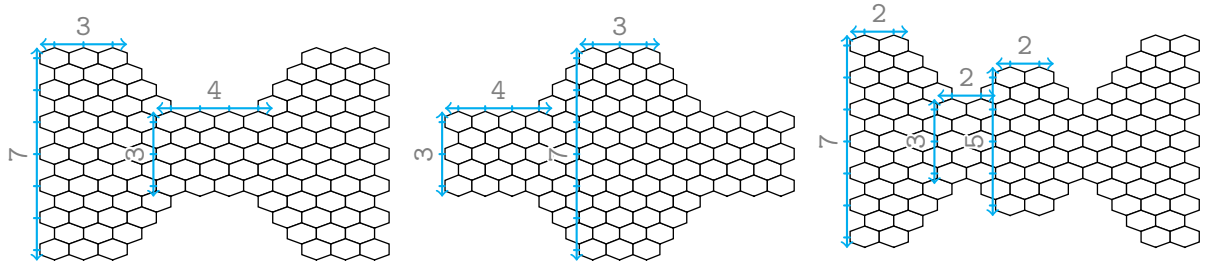
3.2.1. Creating random GNR devices

Recall the three classes of GNR device shapes defined by Jiang et al. [31]: the "butterfly", the "camel", and the "double butterfly" shapes. See Figure 3.6 for examples. Note that each shape is defined by the lengths and heights of its bumps and constrictions. In our work, we generalize this approach a little bit.

We conceptualize GNRs as consisting of a series of alternating bumps and constrictions between two ends. While doing so, we always assume symmetry in both the x and y directions. For example, the GNR structure in Figure 2.3c has ends of height 7 and length 2, followed by constrictions of height 3 and length 2, and finally in the center of the device a bump of height 6 and length 2. All of these height and length dimensions are in columns and rows, respectively, as they are going forward.

By looking at GNR devices this way, the only difference between the different classes of shapes mentioned earlier is how many bumps and constrictions the device has, up to an arbitrarily large number. According to this definition, it's relatively straightforward to create random GNR device shapes, simply by choosing a random number of bumps and constrictions, and then choosing random heights and lengths for each.

In this work, we limit ourselves to GNR devices with zero, one, or two bumps, just so they don't get too big and the simulation time of each device too long. For the same reason, we set some maximum sizes for the dimensions of each bump and constriction.



(a) A GNR with a "butterfly" shape. The sizes of the butterfly's features (a constriction between two symmetric ends) are denoted in rows and columns.

(b) A GNR with a "camel" shape. The sizes of the camel's features (a bump between two ends of smaller height) are denoted in rows and columns.

(c) A GNR with a "double butterfly" shape. The sizes of the butterfly's features (a bump surrounded by two constrictions, in turn surrounded by two ends) are denoted in rows and columns.

Figure 3.6: The shapes defined by Jiang et al. [31]. The sizes of each bump and constriction are denoted.

After creating a shape for the GNR like this, there are some more parameters to the device left unspecified: the size and position of the gate contact, the bias voltage at the back contact, and the size of the drain and source contacts. Note that the source and drain contacts have the same size, since they are symmetric. We constrain these parameters a little bit so they don't collide, but choose them randomly otherwise.

Together, all these randomly chosen parameters define a large solution space of possible devices. When considering all possible values for each of these independent parameters, the size of this solution space quickly explodes. Between the most extreme values for each parameter, this size is on the order of 10^{14} possible devices, which would be entirely too big to exhaustively search. Note that this size of the solution space is orders of magnitude bigger than the amount of devices the evolutionary algorithm will actually check, which is 27 000 for our parameters.

By defining our shapes in this way, the geometry of each device is defined by several more or less independent parameters. When we want to apply a small change, we can simply increment or decrement one of these parameters. This way, we've defined a simple way to apply small mutations to our GNR devices. If we only apply one of these mutations at a time, that's equivalent to only moving in orthogonal directions along the fitness landscape, which might be too limiting. By applying a random amount of these single mutations, we can take incremental steps along the fitness landscape in many directions.

Having defined a way of creating and mutating our GNR devices, let's look at the way we evaluate them: the fitness function.

3.3. The Fitness Function

The fitness function is what tells our evolutionary algorithm what kind of GNR device we are looking for. In this way, it connects the evolutionary algorithm to the ADC circuit requirements. If we were looking for a different GNR device for a different purpose, we could design a different fitness function to do that. In this section we discuss the ADC design fitness function.

The first thing we need to do to evaluate a GNR device's suitability for our purposes is to see how it behaves. Using the simulation model discussed earlier, we numerically determine the behavior of the device under different gate voltages V_G. By doing this we construct a conduction map which maps the gate voltage to the drain-to-source conductance of the device.

This conduction map describes the behavior of our device as a response to a changing input voltage on the circuit. We can use this conduction map to determine the transfer function of the circuit later on, but we will need both the PUN and PDN devices for that. To be able to determine the suitability of a single device without its corresponding complementary device, we instead judge the conduction map itself.

To judge the conduction maps of our devices, we define a set of independent, relatively simple metrics. Each of these metrics corresponds to a different characteristic we want to see in the conduction map of our GNR device. In this section, we go over each of these metrics one by one. We discuss why we need our devices to have these characteristics, how we measure whether they do, and what a device might look like which scores well on these metrics.

With all of these metrics calculated for a device, we determine their weighted average to determine our total device quality, which functions as the device's fitness within the evolutionary algorithm.

3.3.1. Frequency Match

As we mentioned before, one of the features that stand out in the desired behavior of our ADC circuit is the periodicity of each transfer function. For each bit, there's a certain stretch of the circuit's input

value's range which repeats. The length of this stretch—the period—is different for each bit. For the first metric of our fitness function, we will be looking at this periodicity.

The transfer function of each bit behaves like a periodic function with a period of $25 \cdot 2^n\,\mathrm{mV}$, where n is the bit number. This is visible in Figure 3.1. The desired conductance of each GNR devices is also periodic with the same period, as one can observe in Figure 3.3.

For this metric, we treat the conduction map of the GNR device as we would any periodic function. To determine the main frequency of the conduction map, we apply the *discrete Fourier transform* (DFT) to it with respect to the input voltage. This produces a spectral representation of the conduction map, consisting of different amplitudes at each possible frequency. Before performing this transformation, we subtract the mean of the signal to filter out the zero-frequency component which would otherwise always be dominant. We also normalize the signal such that the peak frequency has an amplitude of 1. We then take this frequency decomposition and use it to determine the peak frequency of the conduction map signal.

Since we know the desired period of repetition for each conduction map, we can convert that to a desired frequency of $5 \cdot 2^n\,\mathrm{V}^{-1}$ for each bit with number n. Note that the unit of frequency is V^{-1}, or the reciprocal volt. We can normalize these numbers to be a little more clear by dividing the frequency by the total voltage range. After all, we care about the periodicity of the conductance map *with respect to* the voltage range. After normalization, a frequency of 1 (with no unit) refers to a conduction map which oscillates exactly once over its entire range. The resulting normalized target frequencies end up being 2^{3-n}.

Given this desired frequency, we can score each GNR device based on how close the peak frequency in their conduction map corresponds to it. The frequency match metric is higher the closer the peak frequency is to this target frequency.

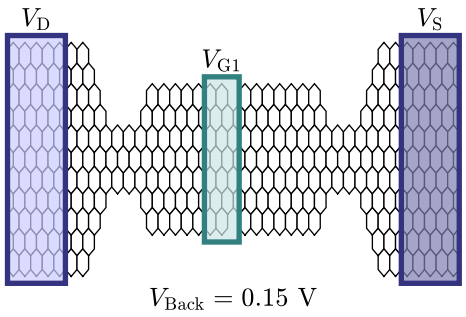
The peak frequency of a signal is not the full story, though. Two signals can have the same peak frequency and one can still be much better suited for our purposes. Let's move on to our second metric, which is closely related to the first: the frequency *mismatch*.

3.3.2. Frequency Mismatch

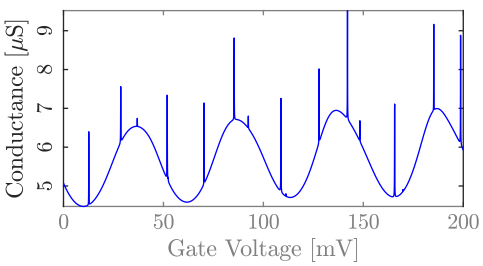
Two GNR devices can have conduction maps with the same peak frequency, and still be very differently suited to our needs. Figures 3.7 and 3.8 demonstrate this by presenting two GNR devices. For each device, the device structure is presented (Figures 3.7a and 3.8a), as well as the conduction map (Figures 3.7b and 3.8b) and the frequency decompositon of that conduction map (Figures 3.7c and 3.8c).

In the frequency plots, we can see that both devices have a (normalized) peak frequency of 4, but they have very different conduction maps. Specifically, the GNR device in Figure 3.7 has a single sharp peak at frequency 4 and the rest of its spectrum is quite small. The device in Figure 3.8, on the other hand, has a much higher amplitude at other frequencies. As a result, the first device's conduction map behaves much more like an ideal sinusoid, whereas the second device's conduction map has a much messier shape.

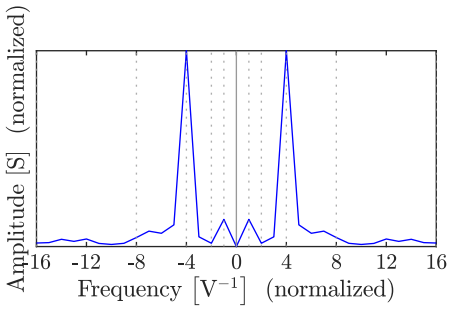
To express this difference as a metric, we determine the contribution to the spectrum of all off-peak frequencies. To do this, we remove the peak frequency from the frequency plot, and take the RMS value of what's left of the spectrum. The higher this number is, the more unwanted frequency components are present, and the less suited the device is to our purpose.



(a) The device's structure.

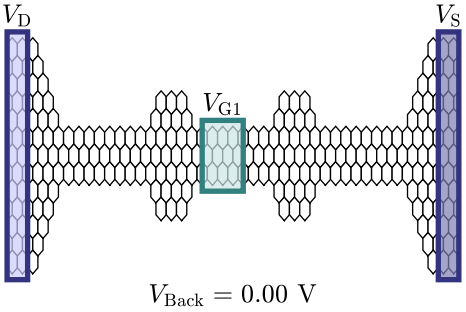


(b) The device's conduction map.

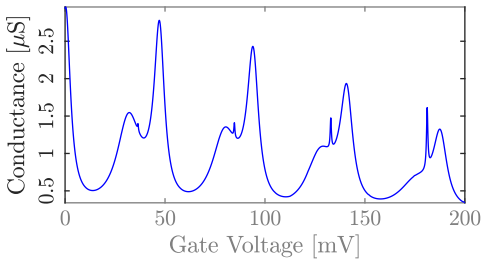


(c) The device's conduction map's frequency composition. Note the sharp peak at 4.

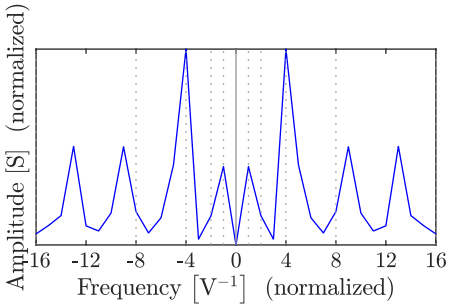
Figure 3.7: A GNR device with a very sharp peak frequency.



(a) The device's structure.



(b) The device's conduction map.



(c) The device's conduction map's frequency composition. Note that the peak frequency is the same, but it accounts for a smaller portion of the total spectrum.

Figure 3.8: A GNR device with a more scattered frequency spectrum.

3.3.3. Phase Match

The difference between the required PUN and PDN devices in our ADC is that they are π `rad` out of phase with each other. This phase is actually quite important, because the matching of the two signals determines the point at which the logic output value switches from `true` to `false` or vice versa. As explained before, it's important that the PUN and PDN match to prevent short circuit states and ambiguous output values. The phase of the signal determines how well these devices match up.

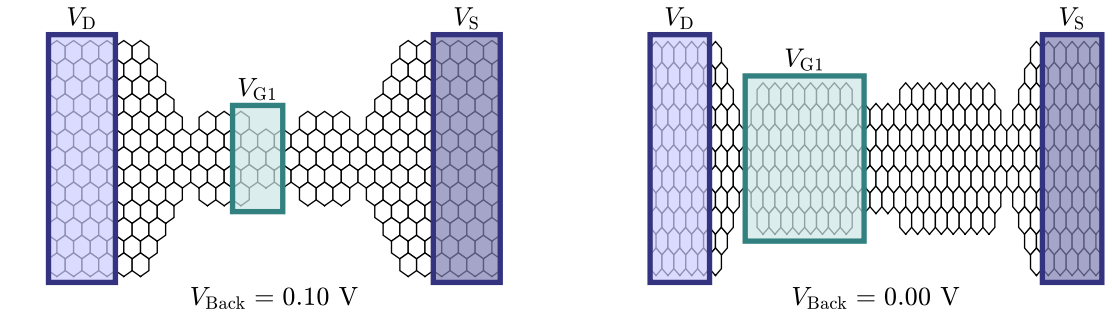
We define the phase required by the PDN to be `0` π `rad`, where at V_G = `0 V`, the conduction value is high. The phase required by the PUN is then $\pm\pi$ `rad`.

Since the different quality measures need to be more or less independent, the phase match score cannot depend on what the peak frequency of the device is. Instead, the dominant frequency of the device's conduction map is determined first, and then the phase of that frequency component is determined. This phase is measured by correlating the conduction map with a wave following the function $G(V_G) = e^{i\omega V_G}$—where ω represents that dominant frequency—and taking the phase of this correlation.

Figure 3.9 presents a pair of devices with a normalized frequency around 4. The conduction maps are presented in the same plot in Figure 3.9c, with a red line representing the approximate logic output value. This value is high when the conductance of the PUN device is high and the conductance of the PDN device is low, and this value is low when the opposite is true. In all other circumstances, it is in between, in the ambiguous region. Since we want this metric to be independent of signal amplitude, the relative thresholds are used when determining when the conductance values are high or low.

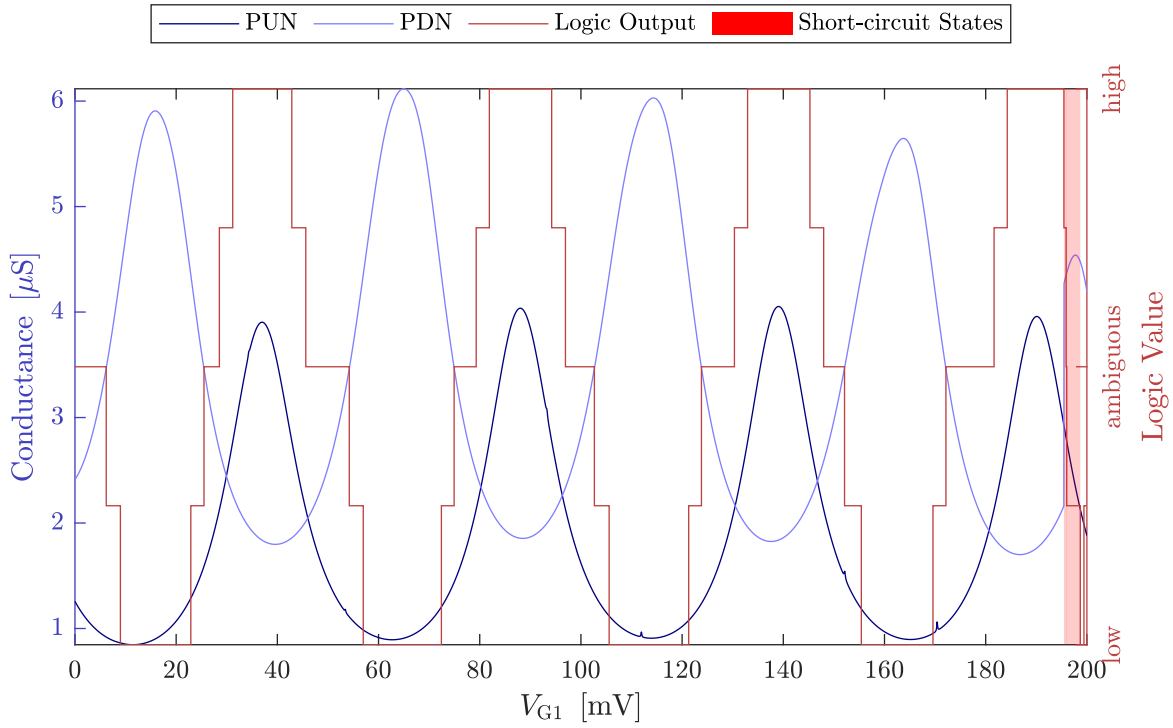
Even though the amplitudes of the conduction maps aren't necessarily very well matched, they have a good correspondence of phase. As a result, there are relatively little areas of the V_G range where the logic output is ambiguous or short-circuited. Because of this, these devices both score high for the phase match metric.

Figure 3.10 presents a pair of devices that are matched much worse. Note that the PUN device (Figure 3.10a) is the same as in Figure 3.9a, it is the PDN (Figure 3.10b) that is poorly matched to it. As a result, only very small sections of the input voltage (V_G) range result in an unambiguous logic value. There are also many values of the input voltage which result in a short-circuit condition, indicated in red. Therefore, this PDN device has a low phase match score.



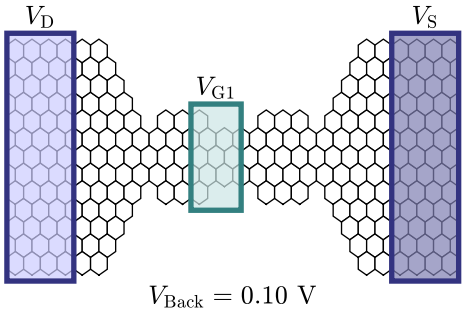
(a) The PUN device.

(b) The PDN device.

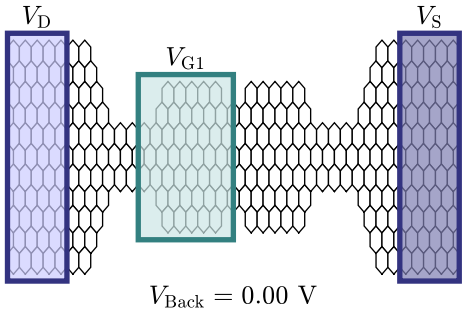


(c) The devices' respective conduction maps, and the resulting logic output value. Note that the logic value is ambiguous when it is neither high now low. Those segments of the transfer function where both devices have a high conductance (resulting in a short-circuit) are marked in red.

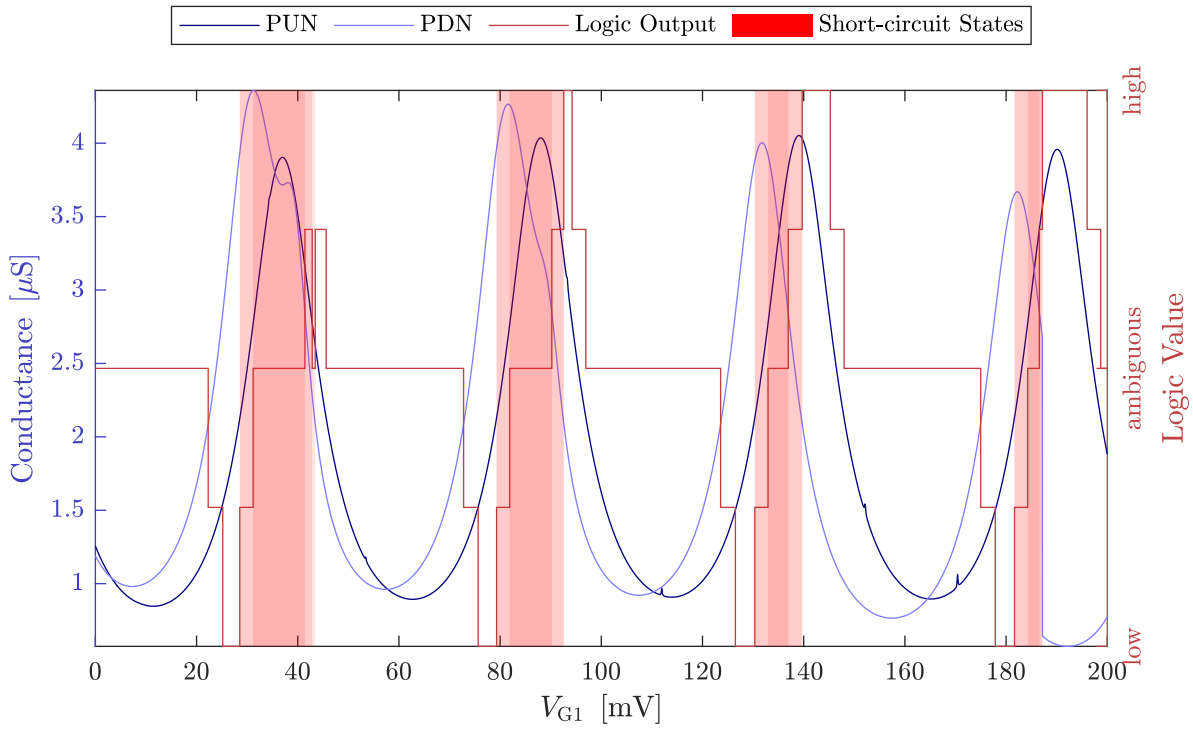
Figure 3.9: Two GNR devices whose phases closely match.



(a) The PUN device. This is the same device as in Figure 3.9a.



(b) The PDN device.



(c) The devices' respective conduction maps, and the resulting logic output value.

Figure 3.10: Two GNR devices whose phases don't closely match, resulting in large portions of the logic output being ambiguous, or creating a short-circuit.

3.3.4. Amplitude Variance

The conduction map of the GNR devices needs to be periodic. It is important that the behavior of the GNR is the same in each period. To ensure this, we introduce some metrics to measure the regularity of each period.

The first of these is the amplitude variance. Each period should have the same amplitude, and the signal should vary as little as possible from this amplitude. In other words, the variance in the amplitude of the conduction map should be as small as possible.

This measurement, again, should be independent of the device's frequency. The first step is to split up the conduction map into different parts, representing the portions of the conduction map where the conductivity of the device is "low", "high", or neither. This is done by simply taking two thresholds, and splitting the signal into the parts where it's above the upper threshold, below the lower threshold, or in between the two. This way, regardless of the specific frequency distribution of the signal, something meaningful can be said about the different periods.

These thresholds are decided as mentioned before, either relative to the conduction map of the device in question, or in absolute terms relative to the conductance of all possible devices. This results in both a "relative" and an "absolute" amplitude variance metric, which are related, but not closely correlated.

After defining these thresholds and splitting the conduction map into periods depending on where they are in relation to those thresholds, we need to define some reasonable measure of the amplitude variance of each of these periods. To do so, we take the minimum value of each "low" period and determine the mean absolute deviation between those minima. We then do the same for the maximum values of the "high" periods, and take the average between these two deviations, weighted to the number of periods that exist of each.

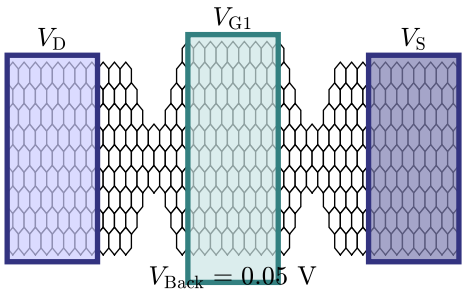
To illustrate this metric, Figures 3.11 and 3.12 present two differet GNR devices. The structure of each GNR device is displayed in Figures 3.11a and 3.12a, and the conduction map of each is plotted in respectively Figures 3.11b and 3.12b. In the case of both devices, the conduction map can be divided into four periods.

The device in Figure 3.11 has a very low amplitude variance. That is, each valley in the conduction map reaches around the same minimum value. Each peak, too, reaches around the same maximum. There is slightly more variance in the maxima of the peaks than there is in the minima of the valleys. The resulting amplitude variance is very low.

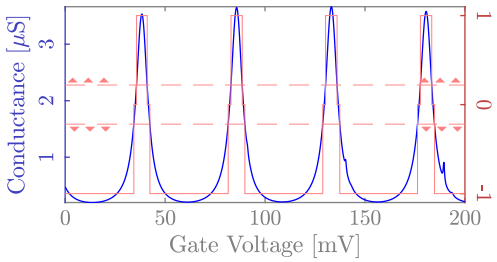
On the other hand, the device in Figure 3.12 has quite a bit more variance in its amplitude. The low points don't differ that much, though still more than is the case for the device in Figure 3.11. Especially the peaks, however, vary quite dramatically between the different periods. As a result, the amplitude variance is a lot higher.

Just to illustrate the difference, the amplitude variance is calculated in relative terms for the device in Figure 3.11 but it is calculated relative to the absolute conduction thresholds for the device in Figure 3.12. In the case of the relative amplitude variance in of the device in Figure 3.11, the relative thresholds are drawn around the median conductance value of the conduction map, and it is easy to see when the conduction values are low or high relative to them. Drawn in the same color as these thresholds is the resulting quantized value, which either has a high, a low, or an intermediate value.

Similarly, the absolute conductance thresholds are drawn in the plot in Figure 3.12b, where the resulting quantized conductance value is plotted in the same color. The low values of the conduction map don't always reach the lower threshold, so these low periods aren't recognized as such. Similarly,

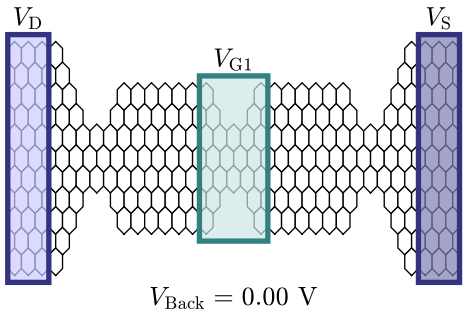


(a) The device's structure.

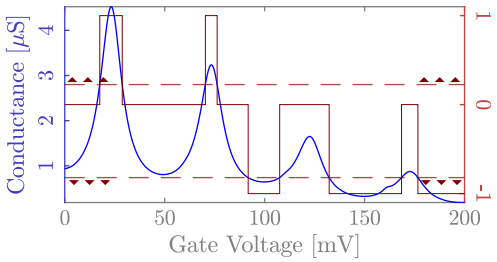


(b) The device's conduction map.

Figure 3.11: A GNR device with a low relative amplitude variance.



(a) The device's structure.



(b) The device's conduction map.

Figure 3.12: A GNR device with a high absolute amplitude variance.

the peaks around a gate voltage of 120 mV and 175 mV don't surpass the upper absolute threshold, and therefore aren't recognized as high conductance values in an absolute sense.

3.3.5. Period Variance

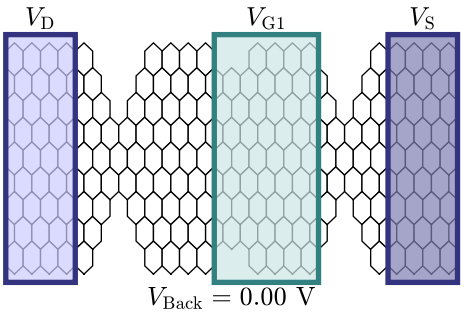
Similarly to the amplitude variance, it's important that each period in either high or low conductance modes is more or less of the same length. Length, that is, in terms of the range of values of the gate voltage that correspond to that period with the same high or low conductance value. Again, we attempt to measure the variance in the lengths of each of these periods, which should approach zero.

We do this in much the same way as the amplitude variance. First, we split the conduction map up into segments depending on where certain thresholds are crossed. These thresholds are the same as before, resulting in an "absolute" and a "relative" period variance. For each segment, we determine the length in terms of V_{G}. The period variance is then the mean absolute variance in these lengths.

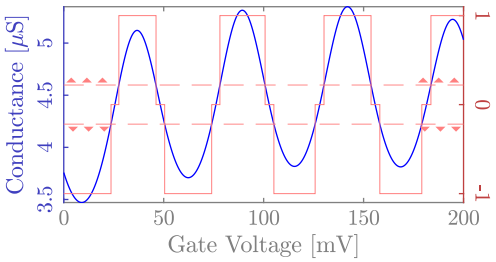
To illustrate the difference, consider Figures 3.13 and 3.14. Once again, Figures 3.13a and 3.14a present a GNR device, the conduction map of which is plotted inFigures 3.13b and 3.14b.

Let's start with the device presented in Figure 3.14. Compared to the absolute conductance thresholds, the conductance value is low for a stretch of over 100 mV, and then there's a narrow peak where the conductance value is high. The lengths of these two periods in terms of gate voltage are very different, leading to a high absolute period variance.

In contrast, the device presented in Figure 3.13 is much more regular. The conduction map consists of several periods of low and high conductance values, each with similar lengths. As such, the relative period variance is low.

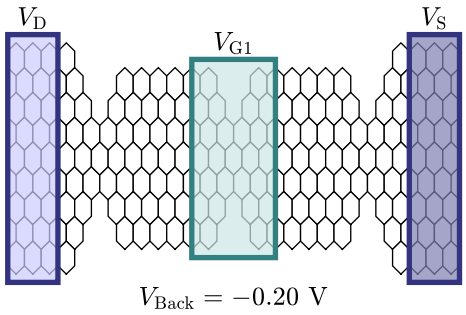


(a) The device's structure.

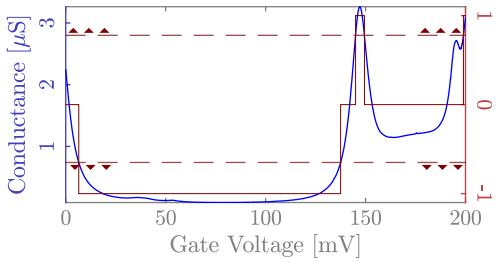


(b) The device's conduction map.

Figure 3.13: A GNR device with a low relative period variance.



(a) The device's structure.



(b) The device's conduction map.

Figure 3.14: A GNR device with a high absolute period variance.

3.3.6. Duty Cycle

So far, we have split the conduction map into segments that are "low", "high", or ambiguous, based on where the conduction is in relation to a set of two thresholds. When the conduction is in between these thresholds, the ADC output value might be unpredictable since there is an insufficient difference between the PUN and PDN conductances. Because of this, we want to minimize the portion of the conduction map in this ambiguous mode.

We also want the portion of the conduction map that is in the "low" conductance mode to be approximately equal to the portion that is in the "high" conductance mode. We introduce a metric called the "duty cycle", borrowed from pulse-width modulators. This duty cycle is defined as:

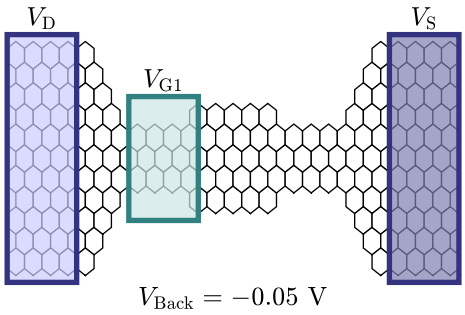
$$\textit{Duty Cycle} = 2 \cdot \min\left(\frac{V_{\text{high}}}{V_{\text{range}}}, \frac{V_{\text{low}}}{V_{\text{range}}}\right), \tag{3.3}$$

where V_{high} and V_{low} are respectively the summed length of the portions of the voltage range of the conduction map where the device is in "high" and "low" conductance mode, and V_{range} is the total length of the conduction map's range.

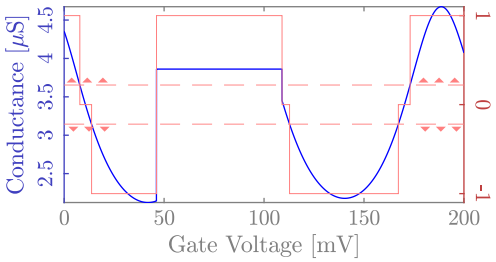
In an ideal device, V_{high} and V_{low} both equal exactly half of the total conduction map, so the duty cycle metric should be as close as possible to 1.

To calculate this duty cycle, we once again split the conduction map into segments depending on either the "absolute" or "relative" conduction value thresholds, resulting in an absolute and a relative duty cycle value. After splitting the conduction map into segments, we can simply take the lengths of those segments to determine the duty cycle metric.

To illustrate this metric, consider Figures 3.15 and 3.16. The device depicted in Figure 3.15 has a high duty cycle score. The conductance has a high value for about the same amount of gate voltage values

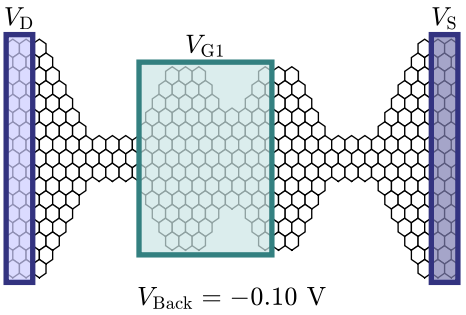


(a) The device's structure.

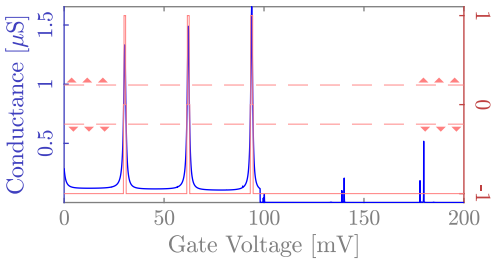


(b) The device's conduction map.

Figure 3.15: A GNR device with a high duty cycle score, meaning the uptime and downtime both approach half the range.



(a) The device's structure.



(b) The device's conduction map.

Figure 3.16: A GNR device with a low duty cycle score, which means the uptime and downtime are badly matched.

as it does a low value. Moreover, the conductance transitions rapidly between high and low values, being at an intermediate value for a small portion of the conduction map.

In contrast, the device depicted in Figure 3.16 has a high conductance value for only a very small fraction of the conduction map, and has a low value otherwise. As such, the duty cycle score is very low.

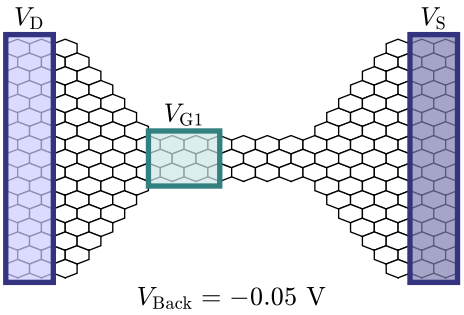
3.3.7. Scale

In order to reliably differentiate between a logic 1 and 0 value, it's important that the high conductance value of the PDN and the low conductance value of the PUN (and vice versa) are as far apart as possible. To ensure this, we introduce a metric which measures the distance between the lowest and highest conductance values in the conduction map.

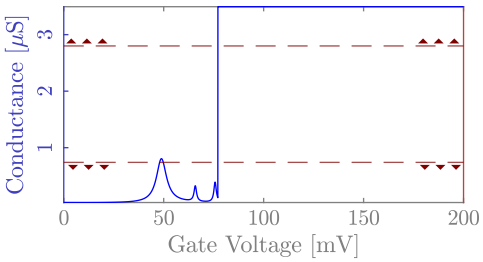
A simple way to do this, would be to take the maximum and minimum values of the conduction map and compare the two. However, those might be outliers that aren't representative of the conduction map as a whole. Rather than taking the most extreme conductance values, we instead take the 20^{th} and 80^{th} percentile values for the conduction map to determine its scale. We then take the logarithm base 10 of the ratio between these two values. We use the logarithmic difference rather than the arithmetic difference, because the logic output of the circuit is determined by the ratio between the conductance values.

We want the scale value to be as high as possible. But we do still want to make sure the PDN and PUN devices are in the same range. A PDN with a very large conduction range is worthless when its highest conduction value is still smaller than the corresponding conduction value of the PUN device.

To encourage overlap, we use the same absolute threshold values as when calculating the absolute

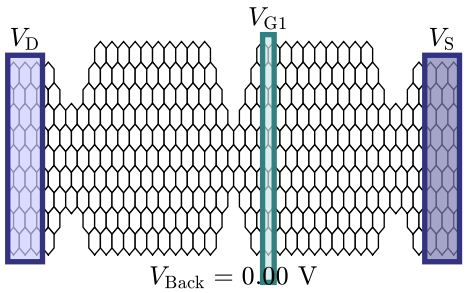


(a) The device's structure.

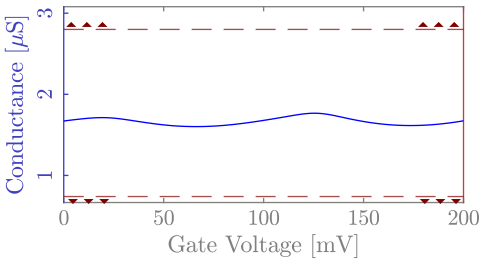


(b) The device's conduction map.

Figure 3.17: A GNR device with a high scale score.



(a) The device's structure.



(b) The device's conduction map.

Figure 3.18: A GNR device with a low scale score.

amplitude variance and such. If the high conduction value of the device is above the upper threshold and the low conductance value is below the lower threshold, the scale score is just the logarithmic difference between the two. If one of the two is in between the thresholds, we multiply this metric by 0.5. If the whole conduction map is above the high threshold or below the low threshold, we set the scale metric to be 0, since the device doesn't match the desired range at all.

To illustrate this, Figures 3.17 and 3.18 present two different GNR devices. To more easily compare the scale of their respective conduction maps (Figures 3.17b and 3.18b), the absolute conductance thresholds of 0.74 µS and 2.8 µS are drawn in red. The conduction map of the device presented in Figure 3.17 spans values from near zero to several microsiemems, whereas that of the device in Figure 3.18 fluctuates in the neighborhood between 1.5 µS and 2 µS, for a much smaller logarithmic range. This difference is reflected in the scale metric.

3.4. Expectations

Due to the principles of evolution, we expect the overall device fitness to increase over time. That is, we expect the average device in the population in a given generation to have a higher fitness score than the average device from the previous generation within that epoch. Note that this does not say anything about the fitnesses of devices of different epochs relative to each other.

We do expect the algorithm to converge in each epoch. By which we mean that in the later generations of an epoch, all devices in the population are similar to each other, with very little difference in their fitness scores. This would mean that the algorithm has reached a local optimum in which no better devices can be found in the near fitness landscape. Subsequent generations would then likely only have very similar devices with similar fitness scores to the previous.

Because each epoch of the algorithm is unrelated to each other, there is no reason why the algorithm can't converge on the same local optimum each epoch. But there is a chance for a new, different local

optimum to be found each epoch. So after a number of epochs, we expect different local optima to be found, where each local optimum might be "found" multiple times. The best of these local optima should hopefully be somewhere close to a global optimum in terms of fitness.

We expect the GNR devices produced by this evolutionary algorithm to score well on the metrics defined in this chapter, and thereby to exhibit the behavior required of them to be used in an ADC circuit. Together, we expect them to be able to implement the required transfer functions of each subcircuit of the ADC.

The ADC circuit consisting of these GNR devices will be much smaller and simpler than conventional ADC designs. As such, it is expected that it will perform significantly better than conventional designs in terms of propagation time, power cost, and circuit area.

4

Results

Now that we know what the evolutionary algorithm looks like, let's see what it's resulted in. Before looking at what the algorithm has produced in terms of devices, we first look at how it behaved throughout the process. We initially look at whether the algorithm indeed tends to generate devices of increasing fitness. But then we're also interested in converging behavior of the process: whether the device pools in each generation tend to become more similar to each other over time: a pool of clones. Whether the difference between generations starts to level off or not. And whether the final states of each epoch are similar to each other.

Knowing that the evolutionary process tends to converge on devices of increasing fitness according to the metrics we've defined, we then look at some of these resulting devices to see how well they match what we expect of them. We see whether the algorithm tends tends towards the type of devices we're looking for and discuss some issues in making a final selection for a best candidate.

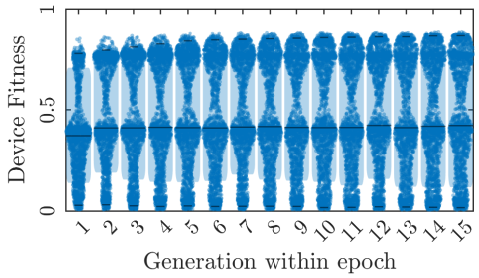
Having found some devices, we then see how well they function together as an ADC. We see how well an individual bit's subcircuit performs, as wel as how accurate the full conversion is. There are several metrics of accuracy and efficiency, for each of which we will discuss how we can determine and score the circuit, so we can then compare them all to the state of the art.

4.1. The Evolutionary Process

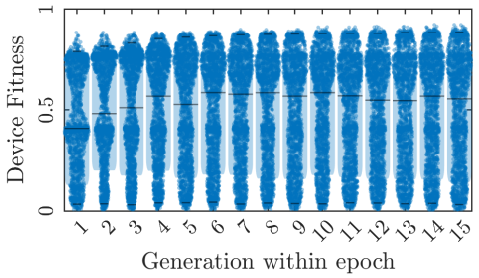
First things first, let's look at how the evolutionary process behaves. There's a few relevant things that we want to know on in that regard.

First of all, we want to know if the process tends to produce devices of increasing fitness, according to our previously defined metrics. This is not to say that we expect each generation to only have devices with higher fitness than all devices from the previous generation. It is, after all, a largely random process. But we do hope and expect that there should be a general upward trend with increasing generations towards higher fitness. Note that each epoch starts over from scratch, so really "generation" here refers to the generation within a certain epoch.

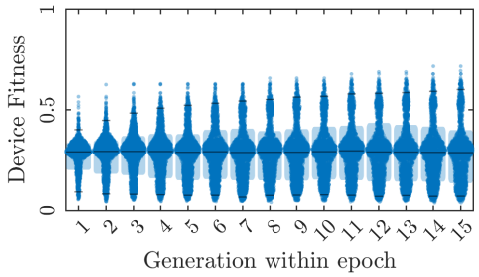
Figure 4.1 demonstrates the growth of the fitness metric over time. Each subplot represents the evolutionary search for one GNR device, where all evaluated devices are marked as blue dots. Each dot is located in the y-direction according to their fitness metric,and in the x-direction according to their generation within the epoch. For visibility, a random scatter in the x-direction is added to each dot according to the density of these dots. This creates the effect of a histogram, where the blue cloud is wider where there are more devices. In each generation, a few relevant percentiles are marked as



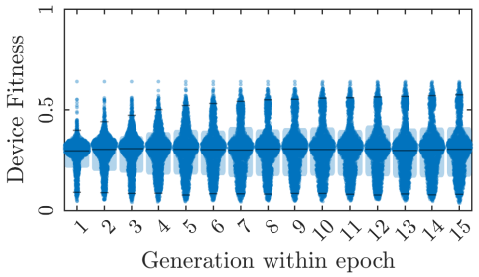
(a) The progression of device fitness through the search for the PDN GNR device of the MSB circuit.



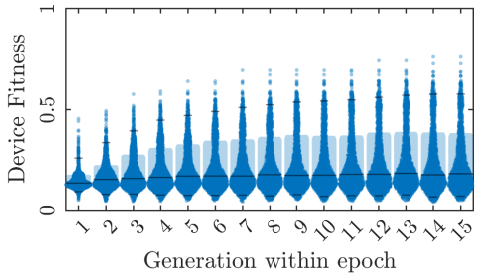
(b) The progression of device fitness through the search for the PUN GNR device of the MSB circuit.



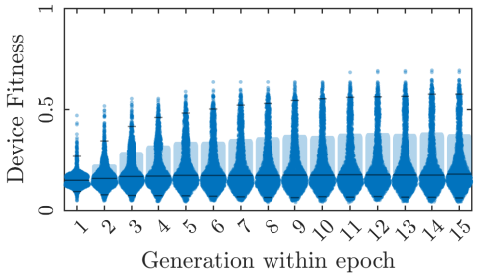
(c) The progression of device fitness through the search for the PDN GNR device of the bit 2 circuit.



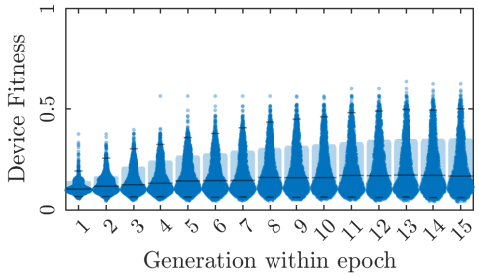
(d) The progression of device fitness through the search for the PUN GNR device of the bit 2 circuit.



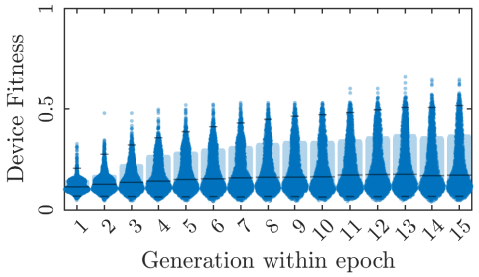
(e) The progression of device fitness through the search for the PDN GNR device of the bit 1 circuit.



(f) The progression of device fitness through the search for the PUN GNR device of the bit 1 circuit.



(g) The progression of device fitness through the search for the PDN GNR device of the LSB circuit.



(h) The progression of device fitness through the search for the PUN GNR device of the LSB circuit.

Figure 4.1: The progression of found device fitness throughout each of the eight search processes.

well. The median—or the 50th percentile—of each collection is marked as a black line. The 16th and 84th percentiles are marked together with a transparent blue box. In a normal distribution, these would represent the one-sigma range. Similarly, a pair of short black lines mark the 2nd and 98th percentiles, which would contain the two-sigma range. These percentile marks make it easier to see trends in these large populations.

We can observe a few interesting things. In all searches, the peak fitness (represented by the 98th percentile mark) increases over time. This is good, because it means the evolutionary process is pushing the fitness up. At the same time we can see that the median fitness of the device pools doesn't change much, indicating that there is maybe too much randomness in the process to efficiently select for high-fitness configurations and prune low-fitness ones.

We can observe a difference in the shape of these data in the searches for the different bits. In the case of the *most significant bit* (MSB), bit 3 (Figures 4.1a and 4.1b), there's essentially a very large spread in fitness from the start. There are two big lumps, or modes, in the population, one at a higher fitness than the other. As the evolutionary process continues, the population around the higher mode grows and the peak fitness increases. But the peak fitness doesn't keep increasing for very long, reaching a plateau after about six generations.

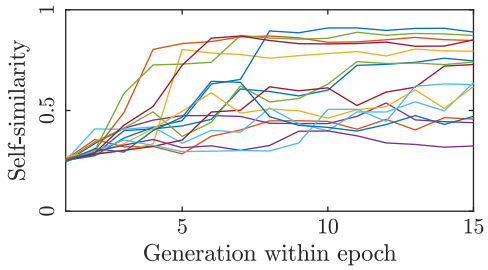
In the case of the other bits, it's more like there's a very wide distribution that gets asymmetrically wider over time. Note that the 84th and 16th percentiles also move outwards from the median. The peak fitness takes a longer time to level off for each next less significant bit. In all cases, the peak fitness levels off at a lower value than for bit the most significant bit, closer to a fitness of `0.5` than to `1`.

Along with the fitness growth over time, we're interested in the converging behavior of the evolutionary algorithm. What we expect is for the device pool in later generations to consist of many similar devices with small differences between them, in contrast to earlier generations which consist of an unrelated pool of devices which can be very different. The reason we expect this is because that would mean that as the process goes on, it converges to variations on a single optimal device or a small set of devices of similar fitness. In evolutionary terms, we expect the ecological niches to be filled by one or a few species with different subspecies, rather than many wildly different species.

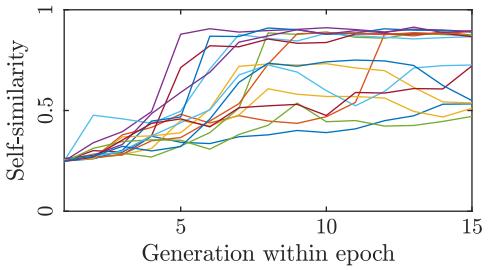
If we were to define some metric of self-similarity, which measures how similar the devices in a population are to each other, we expect the self-similarity of each generation's device pool to follow an increasing trend for this reason.

In order to be able to define measures for the self-similarity of a set of devices, it's helpful to first define some measure of similarity between any two devices. A simple way of doing this is to simply check all numbers in the shape description of both devices and compare them to each other and take the average absolute difference. This would seem to be an appropriate way to do this, since it is related to the mutation mechanism. That is, each mutation attempts to incrementally change one of the attributes of the device shape while leaving the other attributes unchanged as much as possible, so it makes sense to measure the difference between two devices in terms of changes in these attributes. In this way, the difference between two devices is related to the number of mutation steps required to turn one device into the other. The similarity between two devices is then defined as an inverse exponential of the difference, to transform it into a decreasing function between `1` and `0`.

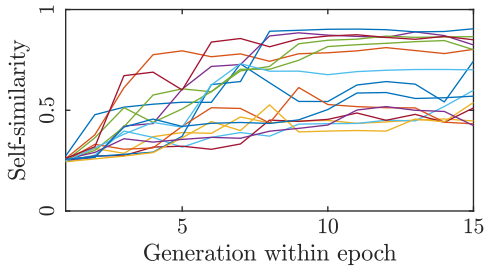
Having defined a metric for the similarity between two devices, the self-similarity of a set of devices can then be defined as the mean similarity between all possible pairs of devices in that set. In the same vein, the similarity between two distinct sets of devices can be defined as the mean similarity between all pairs of devices between the two sets.



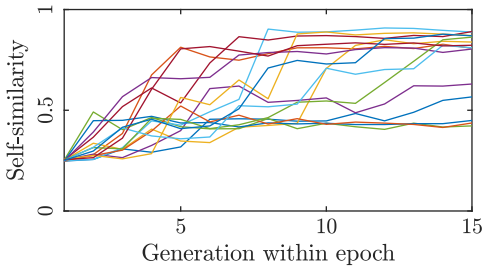
(a) The progression of device pool self-similarity through the search for the PDN GNR device of the MSB circuit.



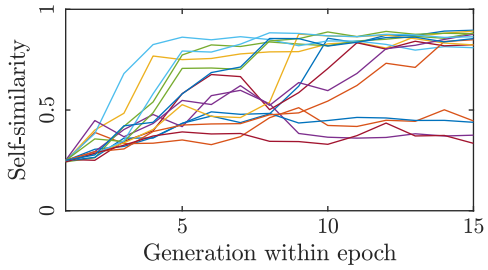
(b) The progression of device pool self-similarity through the search for the PUN GNR device of the MSB circuit.



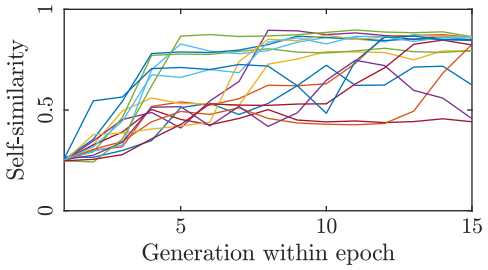
(c) The progression of device pool self-similarity through the search for the PDN GNR device of the bit 2 circuit.



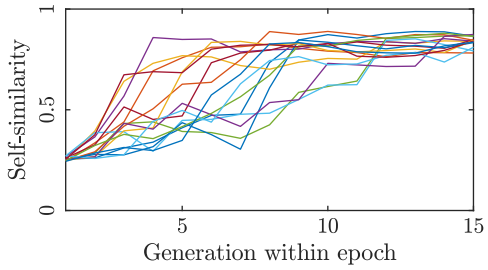
(d) The progression of device pool self-similarity through the search for the PUN GNR device of the bit 2 circuit.



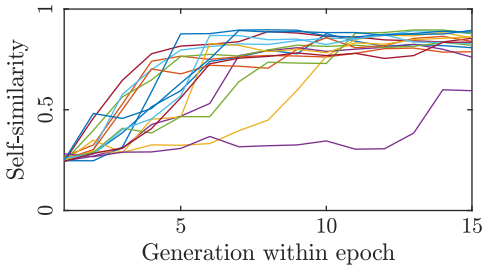
(e) The progression of device pool self-similarity through the search for the PDN GNR device of the bit 1 circuit.



(f) The progression of device pool self-similarity through the search for the PUN GNR device of the bit 1 circuit.



(g) The progression of device pool self-similarity through the search for the PDN GNR device of the LSB circuit.



(h) The progression of device pool self-similarity through the search for the PUN GNR device of the LSB circuit.

Figure 4.2: The self-similarity of devices found throughout each of the eight search processes.

With these metrics defined, we can look at what's happening during the evolutionary process. Figure 4.2 illustrates the self-similarity of each generation of the search. Each plot presents the search for one of the devices of the ADC circuit, where each line represents the self-similarity of the device pools of each generation of one epoch. The color of these lines simply help to distinguish each epoch from one another.

We can observe a few interesting things. Many, but not all, of the lines go up to some value around 1. In other words, not all epochs of the search result in a converging device pool, but most of them do. At the earliest, many searches converge around generation five of an epoch, after which they stay at the same level. In general, once the device pool has converged, the self-similarity doesn't seem to change much, indicating that maybe not so much is happening anymore in terms of exploration of the design-space. What this likely means is that the search has found some local optimum and can't get away from it. By around generation ten, most searches have converged, indicating that the epochs didn't need to be much longer than that.

In the case of the search for the *least significant bit* (LSB) GNR devices, more of the epochs converge than in the other searches. We can interpret this in two different ways. Either there are more local peaks in the LSB device fitness landscape as compared to global trends than there are in the fitness landscapes of the other devices. Or each search is converging quickly on the same peak, which may or may not be the optimum of the entire search space.

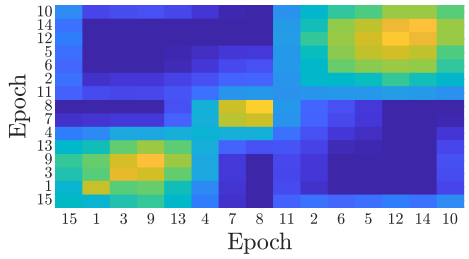
To figure out the difference, it might be good to examine the difference between each epoch. If we look at the similarity between the final generation of each epoch of a certain search, we can see if each epoch is finding a different solution nonlocal with respect to each other, or if they're all finding the same thing.

Figure 4.3 illustrates this comparison. Each colored pixel represents the comparison between two epochs of the search for the same device. These pixels are colored more yellow if the device pools of the two epochs are more similar to each other, and more blue if the pools are more dissimilar. On the diagonal is the self-similarity that we've seen before. The order of the epochs carries no meaning, so they've been shuffled around to cluster epochs with similar device pools together. This results in big yellow squares wherever several epochs each produce vary similar device pools.

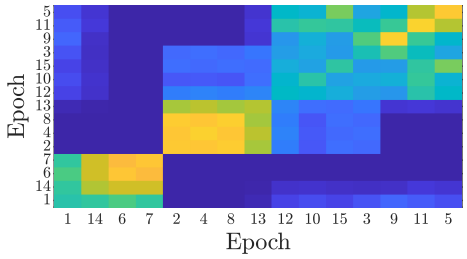
We can observe that the similarity between two epochs correlates strongly to the self-similarity of those epochs. So two sets of devices that have mostly converged will either be very similar or very dissimilar to each other, while sets that haven't converged very well yet will all be kind of similar to each other. We can also see some very clear clustering. That is, we can divide the epochs of each search into about three groups where the datasets produced in each epoch are very similar to each other and dissimilar to the epochs in the other groups. That is, each search has found about three different solutions, where each epoch results in a dataset of devices similar to one of them, such that the epochs can be divided into three groups, where each epoch in a group is very similar to each other. We can say that epochs in such a group ended up in the same neighborhood in the solution space. This holds for all searches. There are larger and smaller groups of epochs who ended up in the same solution neighborhood, indicating that some of these neighborhoods are maybe easier to find. This might imply the existence of further solutions which are even less likely, if the process was run for more epochs.

From all this, we can conclude a couple of things. First of all, the evolutionary algorithm works. There's a trend of increasing fitness, at least at the top of each device pool.

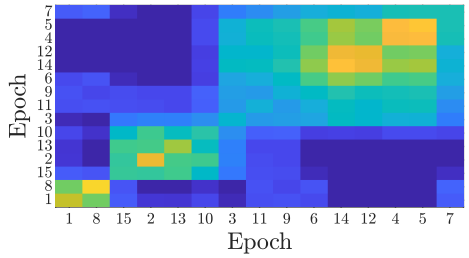
In each epoch of the search, there's a trend of convergence, where all devices in the pool start to become more and more similar and less and less of the solution space is explored. This means the approach of having multiple independent epochs is validated, since different solutions are found in



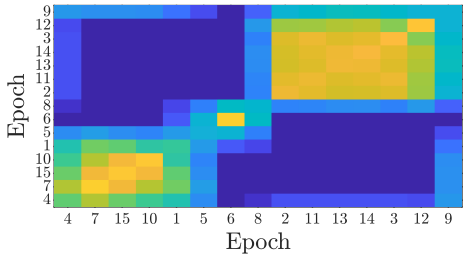
(a) The similarity between the final generation of each epoch of the search for the MSB PDN GNR device.



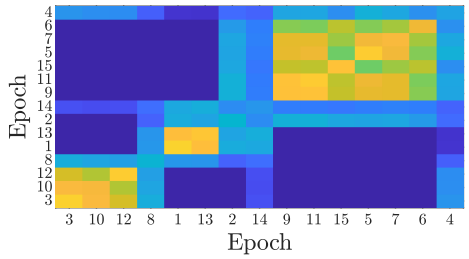
(b) The similarity between the final generation of each epoch of the search for the MSB PUN GNR device.



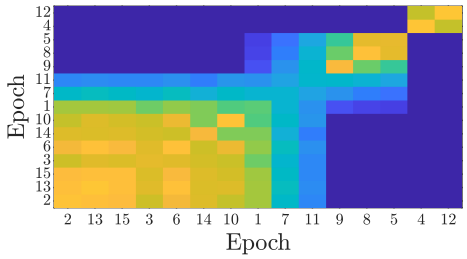
(c) The similarity between the final generation of each epoch of the search for the bit 2 PDN GNR device.



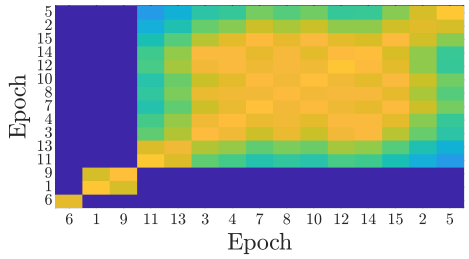
(d) The similarity between the final generation of each epoch of the search for the bit 2 PUN GNR device.



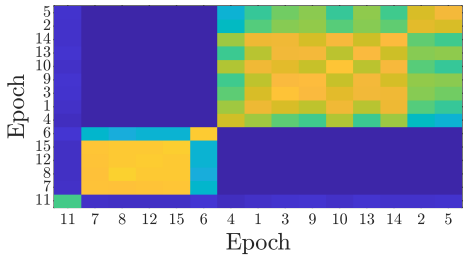
(e) The similarity between the final generation of each epoch of the search for the bit 1 PDN GNR device.



(f) The similarity between the final generation of each epoch of the search for the bit 1 PUN GNR device.



(g) The similarity between the final generation of each epoch of the search for the LSB PDN GNR device.



(h) The similarity between the final generation of each epoch of the search for the LSB PUN GNR device.

Figure 4.3: The similarity between the device pool of the final generation of each epoch.
For each search, the similarity between the device pool of the final generation of each epoch. Blue indicates very dissimilar device pools, with more yellow meaning more similar.

these different epochs.

We can also observe differences between the bits. It seems that the searches for bits of lower significance converge more quickly in terms of self-similarity of the device pool, but converge more slowly in terms of peak device fitness. What this could indicate is that they reach a point where every device in the pool is similar, but there are still many small variations to find the very best device within that neighborhood of the solution space. The final fitness reached for these devices tends to also be lower.

All of this is just from looking at the statistics of the evolutionary process, though. We haven't yet considered what the actual GNR devices produced by these searches look like, or how they perform in a less abstract circuit setting. We'll do that in the next section.

4.2. Found Devices

In the previous section, we've assumed that device fitness is an adequate measure of how well our found GNR devices perform their function within the context of an ADC circuit. In this section we look at how well that assumption holds up. In other words, we consider the devices that the evolutionary algorithm has marked as having a high fitness, and we evaluate whether or not they behave as expected and required.

To illustrate the things we want to talk about, we discuss two different sets of search results of the algorithm: the PDN device of the bit 1 (i.e. the second-least significant bit) subcircuit and the PDN device of bit 3 (i.e. the most significant bit).

The bit 1 PDN device is a good illustration of when the algorithm simply produces a good result. We also use the results for this device to exemplify some of the patterns that emerge in the devices returned by the algorithm.

On the other hand, we make use of the bit 3 PDN device results to illustrate some of the trade-offs that need to be made when the algorithm doesn't necessarily point out the best candidate device on its own. The other six required GNR devices follow from the search results in a similar fashion.

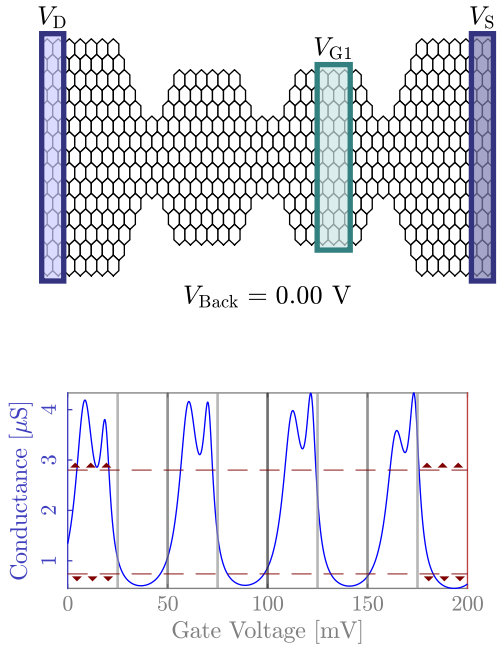
4.2.1. The pulldown device of bit 1 of the ADC

First of all, let's look at what devices the search has identified as potential candidates for the bit 1 pull-down device. The evolutionary algorithm has produced a list of device configurations consisting of the best ones from each epoch. By sorting this list by fitness, we have a shortlist of device configurations which should be quite good.

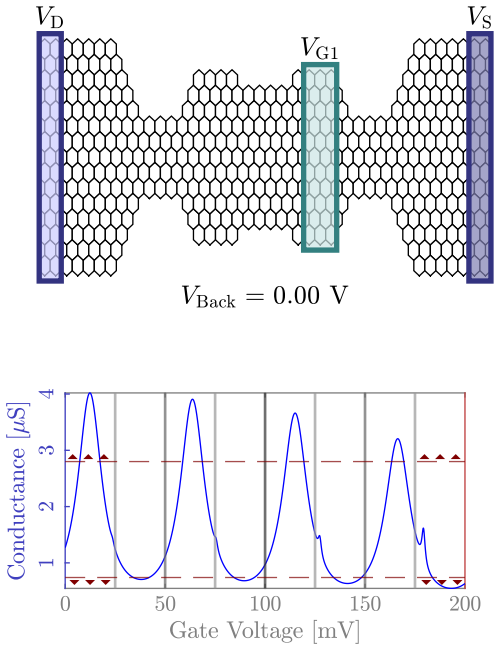
Figure 4.4 presents a sample selection of the device configurations in this list. Each device configuration is accompanied by its conduction map: the device's conductance at each value of the top gate voltage within our specified range of 0 to 200 mV. In each conduction map plot, there's a vertical line separating each eighth part of the input range. The dashed lines mark the two conductance thresholds we've set, where we call any conductance value above the upper threshold "high conductance", while any value below the lower threshold is "low conductance".

These lines denote what behavior we want these devices to have. We want the conductance of the device to be high—the higher the better—in every other segment of one eighth of the input range. At the same time, we want the conductance to be low in the segments in between.

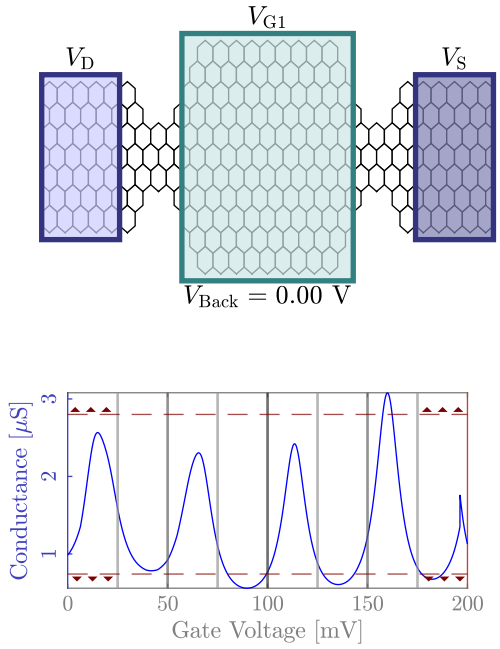
Let's see how they do. There's a few things we can notice from these devices. The first is that they're not very similar in shape. Notably, the number of bumps is different between the different devices, which means it is not possible to get from one device configuration to the other through simple mutations.



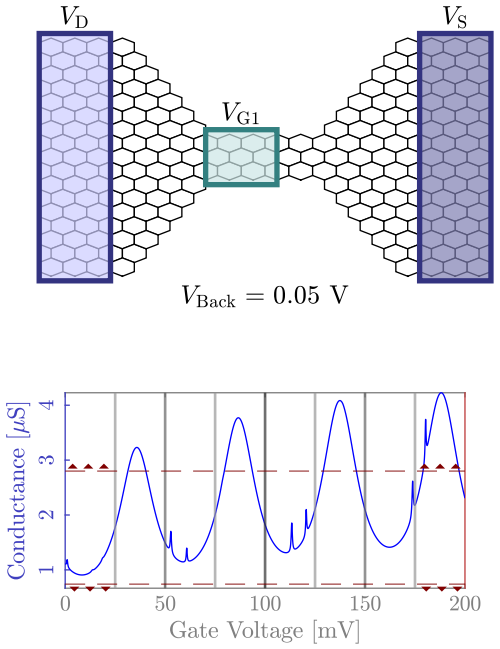
(a) The top ranked device from this search.



(b) The eighth ranked device from this search.



(c) The 21st ranked device from this search.



(d) The 67th ranked device from this search.

Figure 4.4: A selection of candidate devices for the PDN device of the second-least significant bit (bit 1) subcircuit of the ADC. Each device is accompanied by its conduction map, showing the conductance of the device at each value of the gate voltage. The gray vertical lines separate each eighth part of the input signal's range. The dashed lines indicate the defined low-conductance and high-conductance thresholds.

However, Figures 4.4a and 4.4b are quite similar, differing only a little in the pattern of bumps and constrictions, but having the same total dimensions. By inspecting the full list, we can observe that this pattern holds; A handful of different general shapes can be defined, and all devices in the list are quite similar to one of these general shapes. This tells us that the evolutionary process indeed converges towards a small number of locally optimal devices, with local variations which don't change the fitness by a lot. This is consistent with what we observed in the similarity between the end results of different epochs (Figure 4.3).

These locally optimal devices have conduction maps that correspond quite well to what we expect of them. What we specifically mean by that is that the device conductance maps roughly follow a periodic pattern dividing the input range into eight equal areas. Each of these areas alternatingly has a high conductance or a low conductance. Additionally, these high conductance states and low conductance states are clearly distinguishable, at least visually. Or, in other terms, the conductance tends to be above the high threshold in half of the segments and below the low threshold in the other half of the segments. The separation between these segments, or the transition points, also correspond to where we want them to be, i.e. at each eighth of the input voltage range.

These things aren't true for all found devices and they aren't perfectly true for any of them, but the tendencies are clear. We would like the transition points to be more distinct, and to be more consistently at the target input voltages. We would also like the conductance to be more clearly outside of the threshold regions, in order to produce more disparate voltage levels for any subsequent digital logic stages. But generally, the behavior of the devices seems to correspond quite well to what we want of them.

In all, these devices are pretty decent at their job. They're not perfect, but they certainly have potential. Importantly, the top ranked device, Figure 4.4a, seems like it very well might be the best out of all of them. This means that the searching algorithm alone is enough to identify the very best device configuration. Let's see how that plays out for one of the other devices we're looking for.

4.2.2. The pulldown device of bit 3 of the ADC

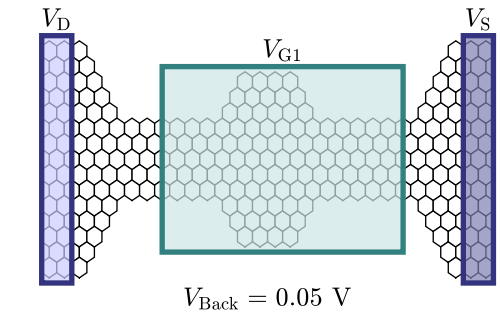
Figure 4.5 presents a sample selection of devices the algorithm has identified as potential candidates in looking for the PDN device of bit 3 of the ADC. Again, the upper and lower conduction thresholds are marked by dashed lines. Here, there is only one transition point that we're looking for, in the middle of the input range. This point is marked by a vertical line.

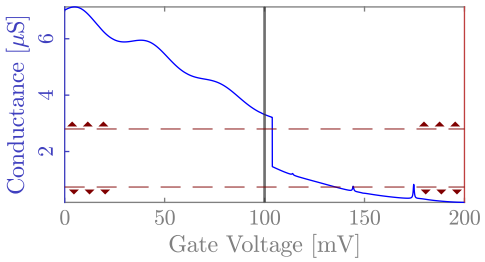
Once again, we notice that the devices aren't necessarily all similar, but they each fall into only a few rough families, and each device is similar to one of those families. In this case, we can identify two types of devices in the figure, with Figure 4.5a similar to Figure 4.5d and Figure 4.5b similar to Figure 4.5c.

All of the devices have a similar conduction map, having a high and decreasing conductance in the lower half of the input voltage range, followed by a discontinuity near the midpoint of the input range, and continuing to decrease slowly in the upper half of the input voltage range.

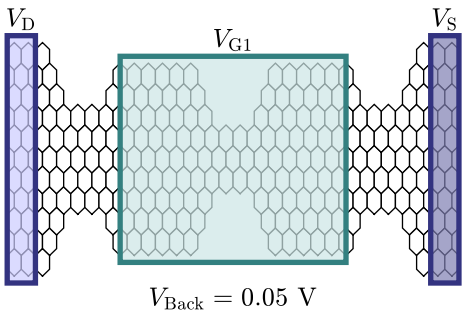
It is not evident, however, that the top ranked device, Figure 4.5a, is the device that's best suited to the job. The discontinuity of the device is at an input voltage a little higher than desired, for one. This discontinuity is also not very large, whereas we would like there to be a large discontinuity separating the high and low conductance states.

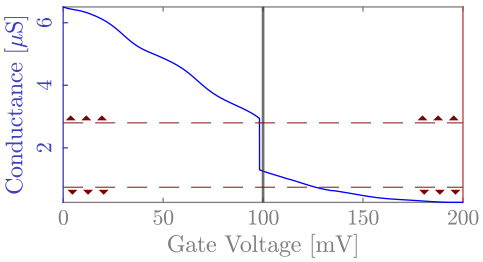
The device in Figure 4.5c has a significantly larger discontinuity right on the midpoint of the input range. Its conductance is quite high beyond that transition point, though, where it ought to be low, not even passing the lower conductance threshold until the input voltage reached 165 mV.



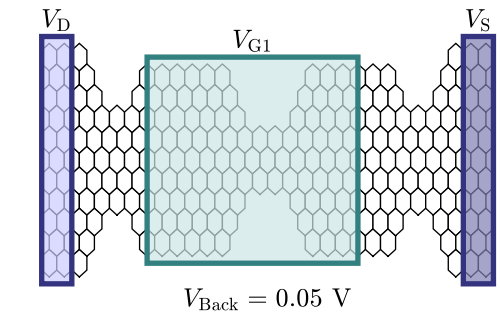


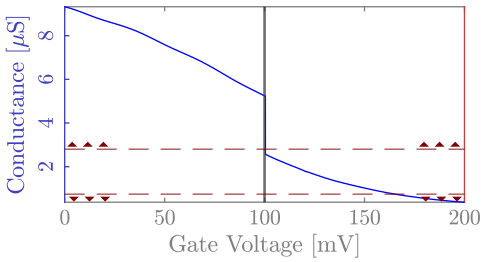
(a) The top ranked device from this search.



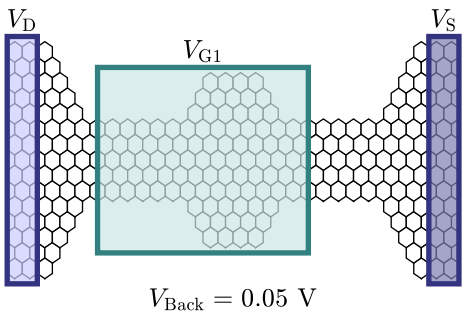


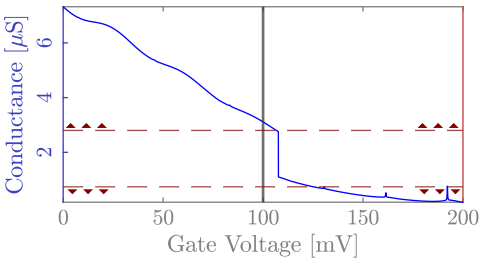
(b) The third ranked device from this search.





(c) The fourth ranked device from this search.





(d) The tenth ranked device from this search.

Figure 4.5: A selection of candidate devices for the PDN device of the most-significant bit (bit 3) subcircuit of the ADC. Each device is accompanied by its conduction map, showing the conductance of the device at each value of the gate voltage. The gray vertical lines separate both halves of the input signal's range. The dashed lines indicate the defined low-conductance threshold and the high-conductance threshold.

Similarly, the device in Figure 4.5d also has some good points and some bad ones. Its conductance is above the upper threshold in the entire segment of the input range before the halfway point. Its conductance is below the lower threshold for a large part of the second half of the input range; the largest part out of these displayed devices, even. But it transitions from high conductance to low conductance quite a bit beyond the halfway point. Before the discontinuity, it is very much in a high conductance state while it should be in a low.

Given these tradeoffs, it is difficult to decide whether the top ranked device is indeed the best one for the job. All found devices are reasonably good, but none is perfect. This makes the matter of picking one out of them nonobvious. This is especially true considering the fact that each pull-down device should accompany a matching pull-up device.

The underlying problem here is that the fitness function utilized in the evolutionary algorithm isn't perfect. There are many things that could be done to more accurately align our device fitness metric to the behavior we want from the device. The weights assigned to each individual metric when combining them into the fitness metric could be tweaked, for example. Or the way that these metrics are combined could be changed, for example to a geometric weighing. Along with that, the searching algorithm could take the simultaneous search for a complementary device into account. For example when evaluating devices at the end of a generation, it could assume the current top result of the complementary search and simulate the joined behavior of the two devices to make a decision. Care would need to be taken to ensure convergence then, however.

Rather than do all that, for now we just make the final selection by hand. The most promising candidates for both the PDN and PUN devices are selected and the joint behavior of each pair is simulated at the circuit level. Out of these pairs, the best pair of devices is selected. This is done by hand again. Some of the considerations when selecting a pair of devices are discussed in the next section.

4.3. The ADC Circuit

Having selected a set of GNR devices to fulfill each role in the ADC circuit, let's now look at what that means for the entire ADC circuit. First we'll look at what a the previously discussed devices look like in their complementary circuits and how they behave. Then we'll assess their behavior when working together as an ADC circuit. Finally we provide a comparison of the proposed ADC with state of the art equivalent counterparts.

As a reminder, Figure 4.6 presents the generic organization of the complementary circuit. The marked PUN component in each of these circuits is implemented by one of the PUN GNR devices found by the search algorithm, and the PDN component by the corresponding PDN GNR device. The output signal of this device, V_{out}, follows Equation (4.1), where $G_{PUN}(V_{in})$, $G_{PDN}(V_{in})$ represent the conductance of, respectively, the PUN and PDN device as a function of V_{in}, and R_{PDN}, R_{PUN} its resistance.

$$V_{out}(V_{in}) = \frac{G_{PUN}(V_{in})}{G_{PUN}(V_{in}) + G_{PDN}(V_{in})} = \frac{R_{PDN}(V_{in})}{R_{PDN}(V_{in}) + R_{PUN}(V_{in})} \tag{4.1}$$

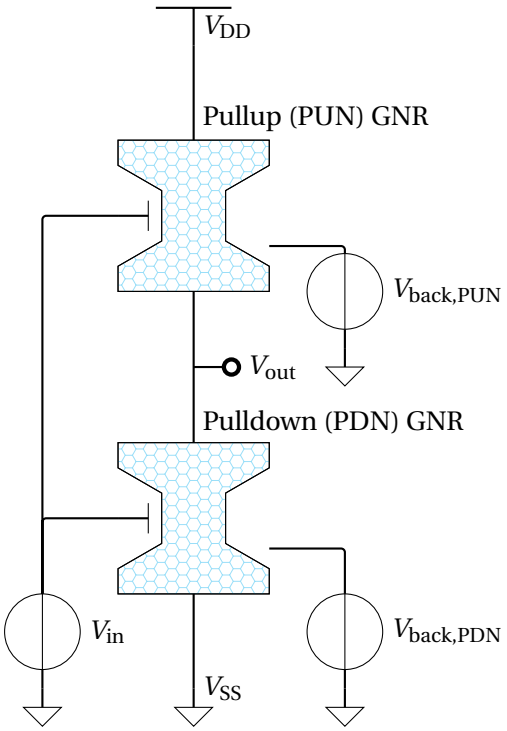


Figure 4.6: The complementary GNR device circuit.

4.3.1. Bit 1 of the ADC

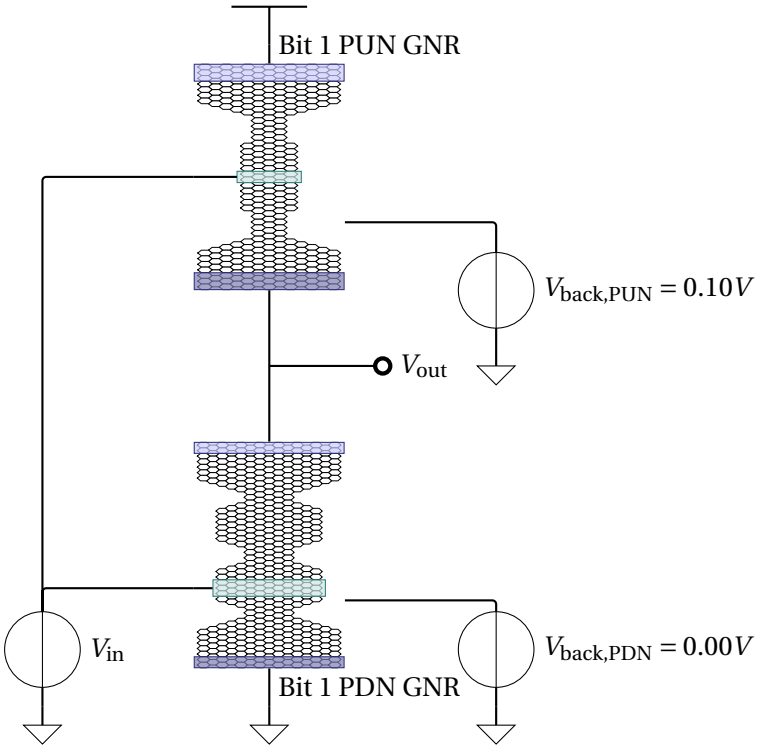
Figure 4.7 presents the implementation of the ADC's bit 1 subcircuit. This circuit converts the analog input signal into bit 1 (i.e. the second-least significant bit) of the binary representation of the value of that input signal. The circuit itself is depicted in Figure 4.7a, which consists of two GNR devices. The pair of devices in the circuit are hand-picked from the top results of the search algorithm for the PUN and PDN devices of bit 1 of the ADC.

Figure 4.7b displays the behavior of this circuit, in the form of a transfer function. That is, it captures the output signal V_{out}, plotted as a function of the input signal and top gate voltage V_{in}. Marked in the plot are a set of vertical lines of different thicknesses separating each eighth part of the input voltage range. A horizontal line marks the halfway point of the output voltage's range. The signal is said to transition between high and low whenever it passes this halfway point appreciably.

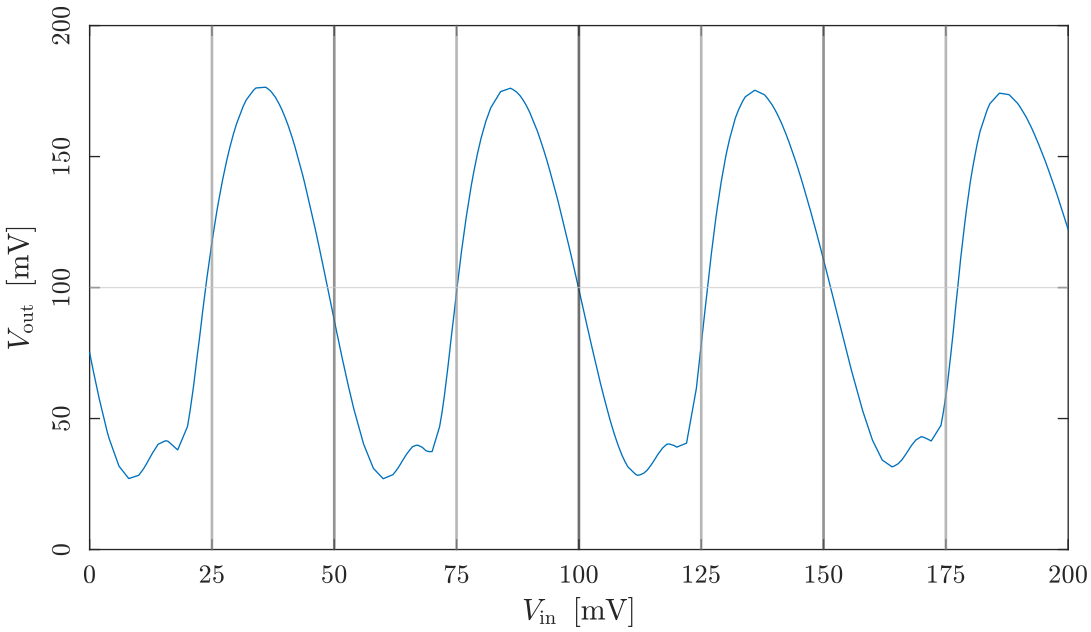
The devices have been selected in such a way that the transition points of the output signal are close to each eighth fraction of the input range. In terms of the image, the signal should cross the horizontal line at the same points where it crosses each vertical line.

If the signal transitions continuously, there is a certain portion of the input signal's range where the output signal is around 100 mV. In that situation, the output signal cannot reliably be determined to be either low or high, and therefore any subsequent digital logic stages won't have a valid input signal. It is because of this that the slope at the transition points would ideally be as steep as possible.

Looking at the plot, we can observe that the circuit performs fairly well in terms of reproducing the expected behavior of ADC bit 1 output. Especially the transitions around input voltage levels 75 mV and 100 mV are very close to where they should be, with the other transitions missing the mark slightly. The slopes at the transitions aren't especially steep, and the signal's amplitude isn't very large. These things could prove problematic in a digital logic stage, which depends on a large difference between a low and a high logic level.



(a) The complementary logic circuit, using the GNR devices found by the algorithm.



(b) SPICE simulation results, plotting output voltage as a function of input voltage. The gray vertical lines separate each eighth part of the input signal's range. A horizontal line marks the halfway point of the output signal's range, separating low and high signal values.

Figure 4.7: The circuit and corresponding SPICE simulation results of the second-least significant bit (bit 1) subcircuit of the ADC.

Before we worry about that, however, let's look at what happened to bit 3.

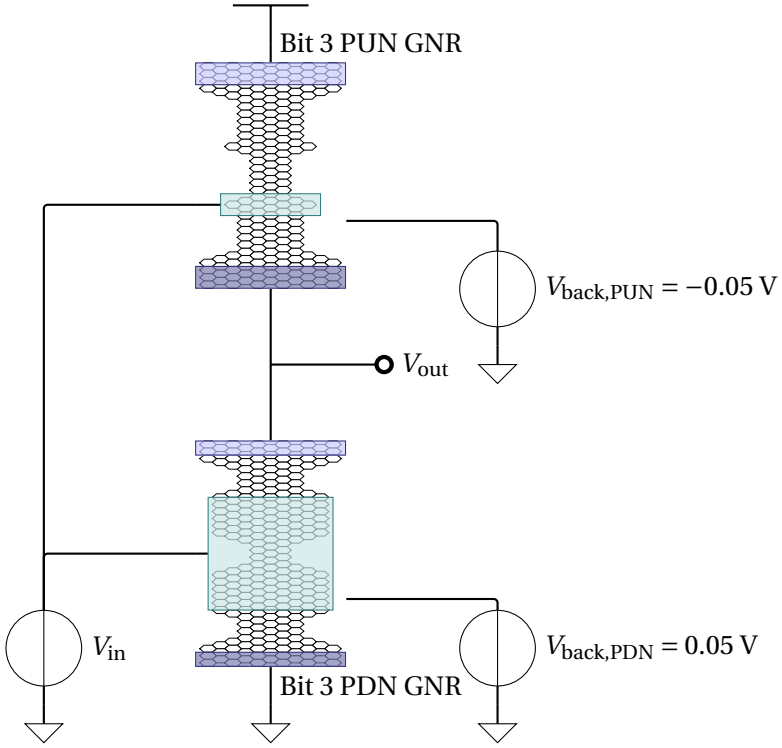
4.3.2. Bit 3 of the ADC

The implementation of the bit 3 subcircuit is depicted in Figure 4.8. This circuit converts the analog input signal into bit 3 (i.e. the most significant bit) of its binary representation. Again, the circuit structure including the geometry of the GNR devices is depicted in Figure 4.8a, and the transfer function in Figure 4.8b. In this case, a vertical line is only drawn at the one transition point we care about, in the middle of the range of the input voltage.

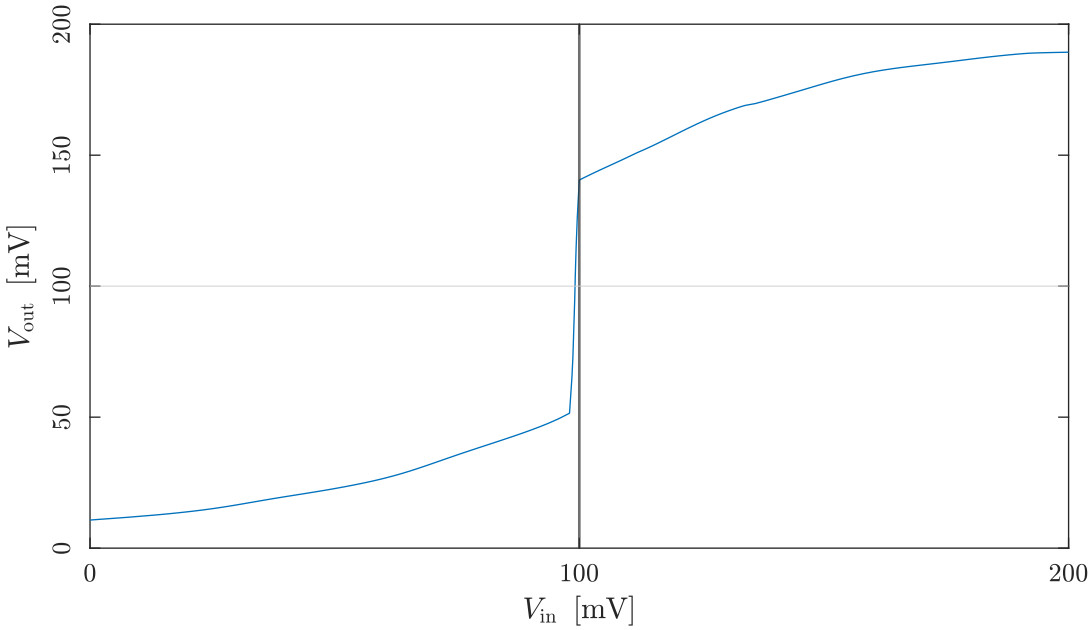
What we would like this transfer function to look like is for it to be low in the left half of the plot and high in the right half, transitioning between the two as quick as possible right in the middle. This is more or less what we observe in the Figure: The signal transitions just slightly before the midpoint. What's expecially noteworthy is the slope with which the signal transitions from low to high, which is almost vertical. It's important to note that the transition is not instantaneous or discontinuous. That is, there are sample points in the data of the curve in between the two levels.

Another thing to note is that the signal increases monotonically. In other words, the higher the input voltage, the higher the output voltage. This holds over the entire observed range of the input signal.

These things together open up some possibilities. Specifically, this circuit could be used as a sort of amplifier, in addition to its functionality as one of the bits of the ADC. Let's take some time to look at this.



(a) The complementary logic circuit, using the GNR devices found by the algorithm.



(b) SPICE simulation results, plotting output voltage as a function of input voltage. The gray vertical lines separate both halves of the input signal's range.

Figure 4.8: The circuit and corresponding SPICE simulation results of the most significant bit (bit 3) subcircuit of the ADC.

4.3.3. Amplification

Suppose we have the circuit discussed in Section 4.3.1: a circuit which converts the input signal to bit 1 of its binary representation. A problem of this circuit is that the output signal doesn't have a very large amplitude. When the output signal is supposed to reach close to the rail voltage, it often doesn't pass the halfway point by very much. This results in significant portions of the input signal's range where it is difficult for a digital logic circuit to determine the bit's value. A consequence of this could be undefined or incorrect behavior.

What we could do to partially remedy this is to amplify the ambiguous signal. We can put a copy of the bit 3 subcircuit discussed in Section 4.3.2 as a second stage behind the first circuit. In other words, we would be feeding the output of the bit 1 subcircuit into the input of the bit 3 subcircuit and taking the output of the latter.

This setup is presented in Figure 4.9. Note that the GNR devices in the first stage are the same as in Figure 4.7a and the GNR devices in the second stage are the same as in Figure 4.8a. We've plotted, in Figure 4.9b the signal after the first stage, labeled "unamplified". The line labeled "amplified" is the signal after the amplifying second stage.

One can observe that the amplification works as expected. When the signal is above the middle of the range, it gets pushed further up, and likewise down when it's below the range middle. This should make it easier for a subsequent stage to determine the intended binary value of the signal.

The bit 1 output similarity with its ideal shape can be further improved by cascading more amplifiers, as indicated in Figure 4.10b, where four stages of such amplifiers are chained after the initial converter circuit. Each subsequent stage is marked by one ">" symbol for each stage of amplification. What we can observe in Figure 4.10b is that with each amplification stage, the signal becomes more like a square wave, shooting from a low value to a high value and staying there until the next transition.

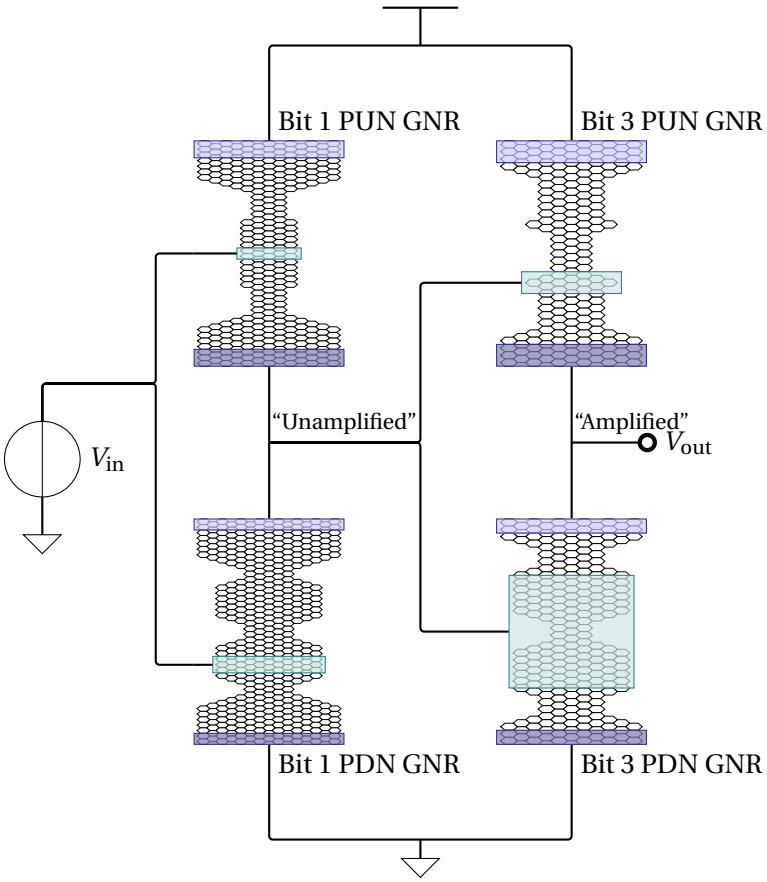
It's not a perfect solution, however. The amplified signal never quite reaches the rail voltages of 0 mV and 200 mV, instead saturating around 10 mV and 190 mV, respectively. This has to do with the transfer function of the amplifier circuit, shown again in Figure 4.11.

In this figure, the diagonal where $V_{in} = V_{out}$ is marked with a dashed line. At three points, the transfer function intersects this diagonal. At these points, the output voltage will be the same as the input voltage, so the signal remains unchanged.

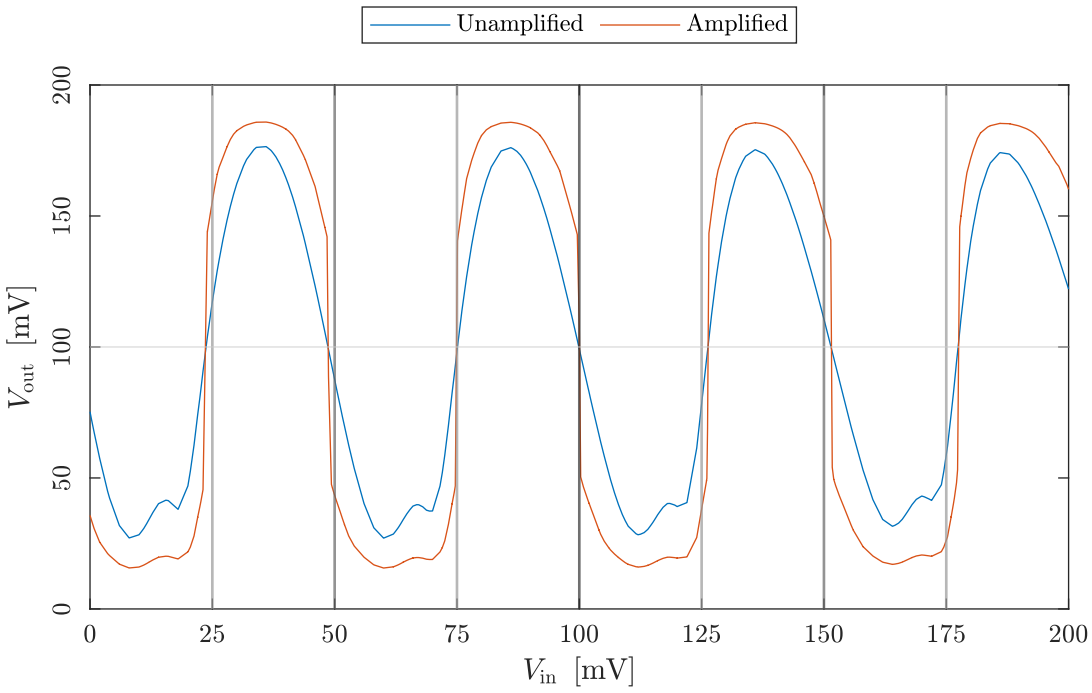
Let's look at what happens near the rightmost of these intersections, around 188 mV. When V_{in} is lower than this intersection point, $V_{out} \geq V_{in}$, so the amplified signal increases with respect to the input signal. When V_{in} is higher than this intersection point, $V_{out} \leq V_{in}$, so the amplified signal decreases with respect to the input signal. As a result, any input signal near this intersection point will move towards the intersection value.

The same things happens at the leftmost intersection, around 13 mV. We can call these points in the transfer function "stable points". The opposite happens in the middle of the range: Unless V_{in} exactly equals the intersection value, the amplified signal drifts *away* from the intersection. Due to this, we call that intersection "metastable".

When multiple of these amplifier circuits are chained together, any signal converges to either of the stable points. This can be observed in the amplified signals in Figure 4.10, where any deviation from the metastable point in the middle of the input range is amplified towards the stable points. The end result resembles a square wave, where the signal only ever assumes the values of either of the stable points.

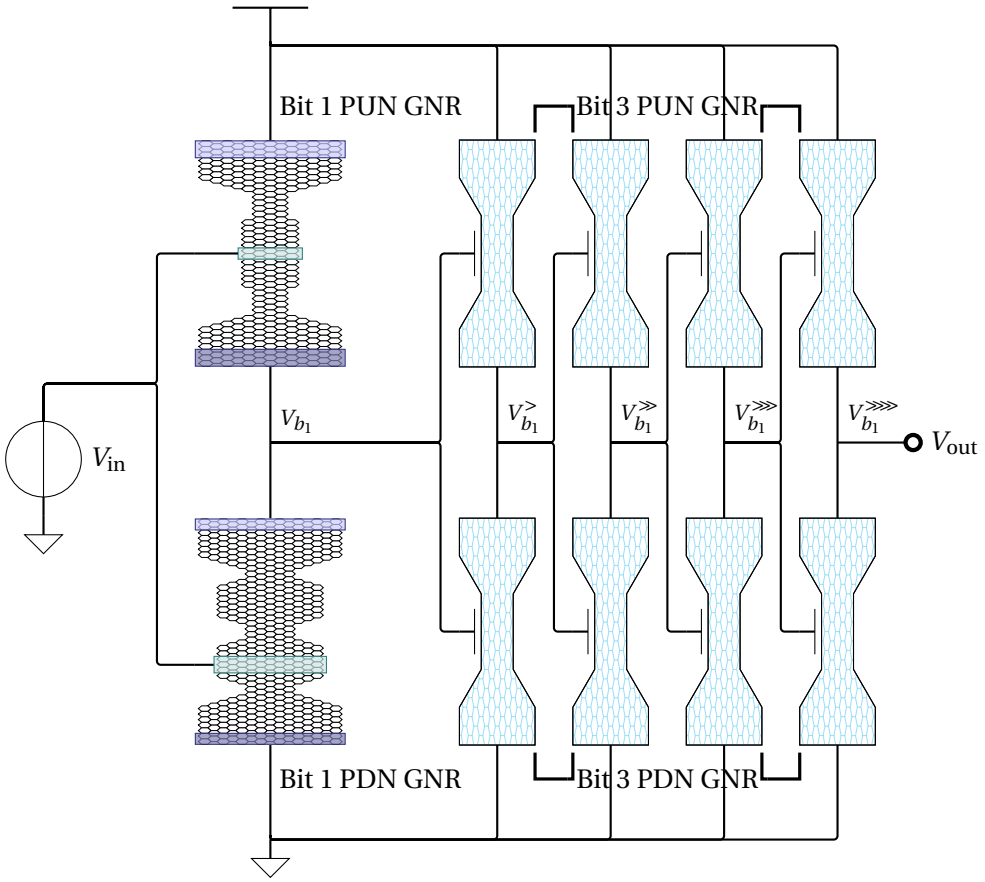


(a) The complementary logic circuit, where the first stage, the functional stage, is the same circuit as seen in Figure 4.7a and the second stage, the amplification stage, is the same circuit as seen in Figure 4.8a.

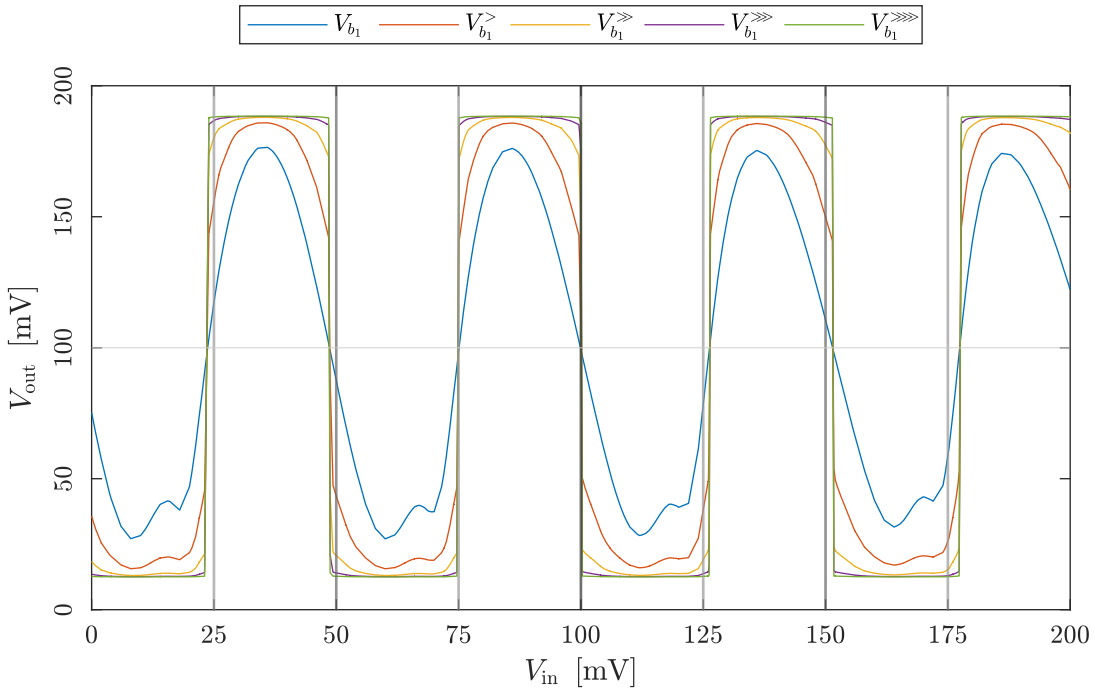


(b) Results of a SPICE simulation of the circuit. The gray vertical lines separate each eighth part of the input signal's range. The curve marked "unamplified" represents the output signal without the amplification stage.

Figure 4.9: The schematic and corresponding SPICE simulation results of a circuit performing the function of the second-least significant bit (bit 1) subcircuit of the ADC, but amplifying the output signal for easier differentiation.



(a) The circuit schematic, with an abstracted representation of the amplification stages.



(b) Results of a SPICE simulation of the circuit. The triangle symbols in the legend denote repeated amplification steps. The gray vertical lines separate each eighth part of the input signal's range.

Figure 4.10: The schematic and corresponding SPICE simulation results of a circuit performing the function of the second-least significant bit (bit 1) subcircuit of the ADC, but amplifying the output signal repeatedly for easier differentiation.

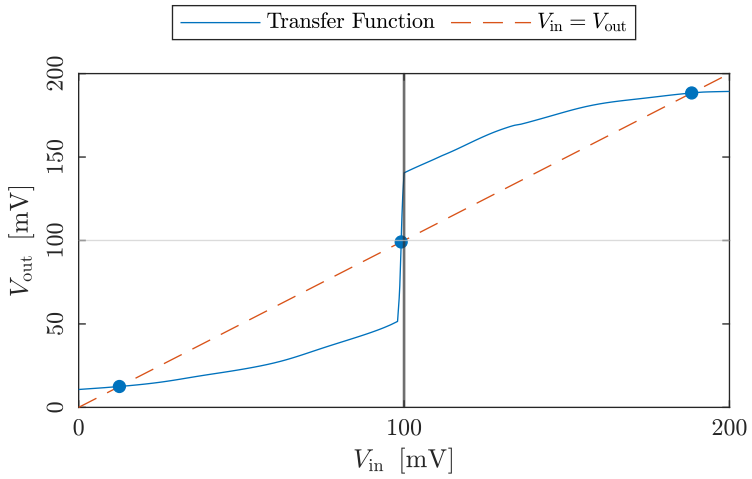


Figure 4.11: A behavioral simulation of the amplification stage, where output voltage V_{out} is plotted as a function of input voltage V_{in}. This is the same figure as in Figure 4.8b. A diagonal line is marked where $V_{out} = V_{in}$ with the intersections of the transfer function marked as dots.

What the amplifier can't change, however, are the signal transition points. For example, the original signal V_{b_1} crosses the middle of the range around 177 mV when we would like that to happen at 175 mV instead. Since the amplifier only increases the signal strength, this transition point remains unchanged with any number of amplifier stages.

Similarly, if there were any noise on the input signal that would cause it to transition at a different point, this would carry over to the amplified signal.

Nevertheless, for the purpose of turning an ambiguous signal into a clear Boolean value, the amplifier circuit performs quite well. Any number of these circuits could be added to the ADC circuit according to the need of any subsequent digital circuitry.

Now let us move on to the full ADC circuit.

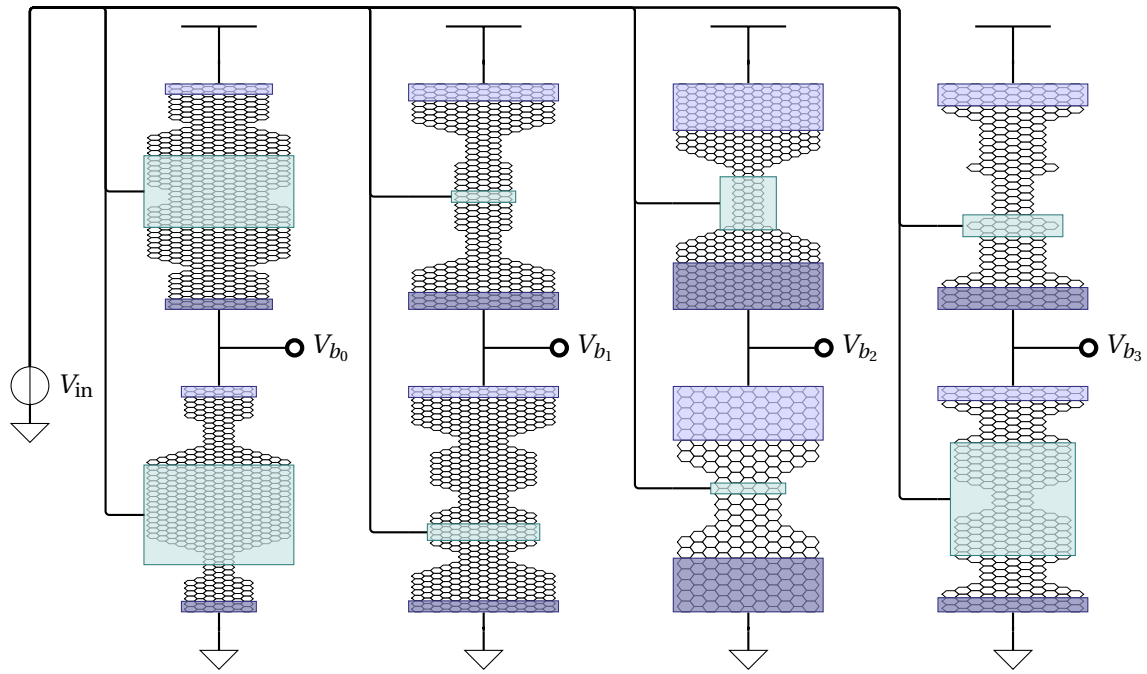
4.3.4. The Full ADC Circuit

When we combine the GNR devices found by the evolutionary algorithm for each bit, we get the full 4-bit ADC circuit depicted in Figure 4.12a. Note that each GNR device in the circuit has a different shape. They also each have a different required back gate voltage V_{Back}, though this is not specified in the figure for the sake of simplicity.

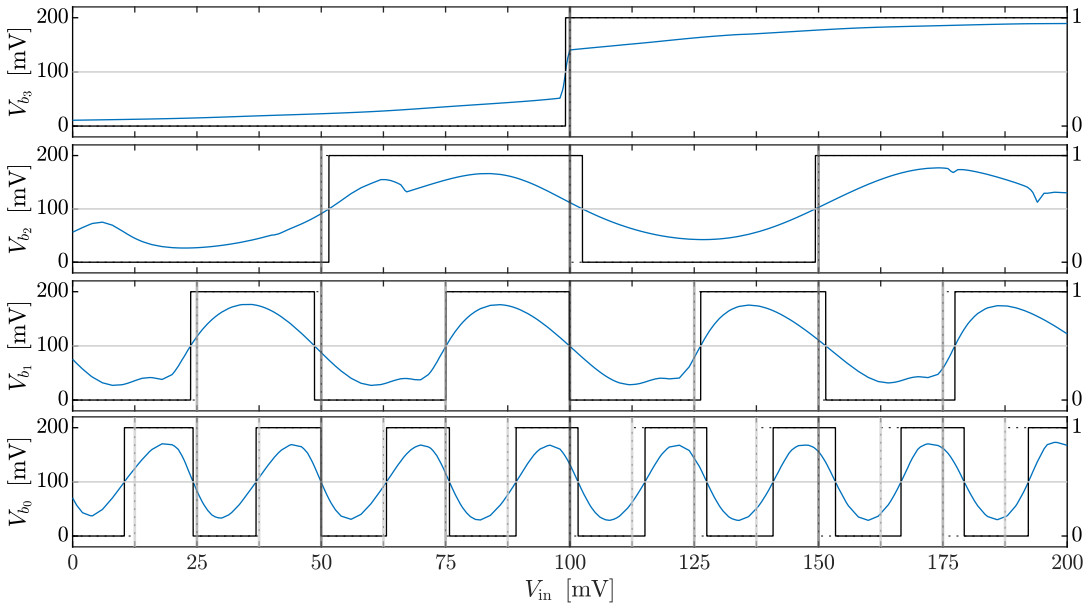
Figure 4.12b displays the results of a SPICE simulation of the circuit. The output voltage signal of each bit is plotted with respect to the input voltage presented at their top gates. In each of the plots, the required transitions are marked with gray vertical lines. A horizontal line is drawn in the middle of the signals' range, which separates high and low output values.

One can observe a few interesting things in these plots. For one, we notice that the transition point locations match up fairly well with their desired positions. The largest discrepancies are in V_{b_0}, and even there only for some of the transitions. We can see there that the signal's transition frequency is just slightly larger than intended, which causes the transition points to drift away for higher V_{in} values.

One way to determine the performance of the ADC's subcircuits is to determine how often the output signal is correct. That is, for which fraction of values of V_{in} does the output signal match up with the



(a) The circuit itself, containing four different GNR devices, each found by the evolutionary algorithm.



(b) A simulation of this circuit plotting each bit's output voltage signal with respect to the input voltage. Dotted lines mark the desired digital value of each signal.

Figure 4.12: The full 4-bit ADC circuit.

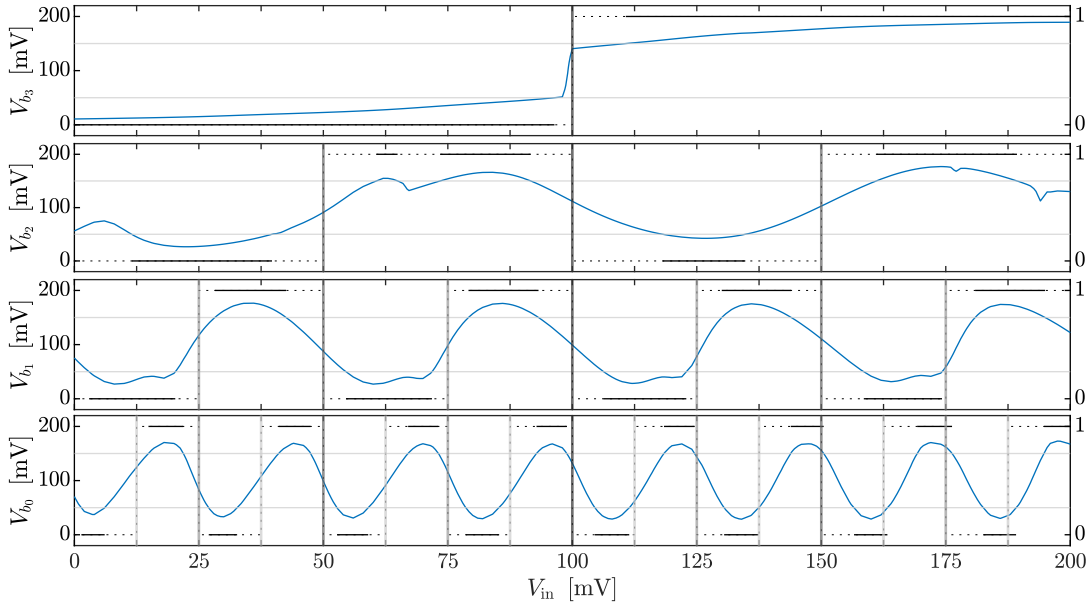


Figure 4.13: A simulation of the full 4-bit ADC circuit, showing each bit's output voltage signal as well as its digital value, with respect to the input voltage.

desired digital signal. Here, we define the digital value of the output signal as a comparison between that signal and the halfway point of its range: 100 mV. So we say that the signal is high if it's above this halfway point and low if it's below.

The required digital value of each output signal is plotted as a dotted line in Figure 4.12b. Bit 3, the most accurate subcircuit, has a correct value for 99.6 % of values of V_{in}. On the other hand, the least accurate one, bit 0, is correct for only 83 % of inputs.

This makes sense, when you consider that for bit 3, there's only one transition. Meanwhile, bit 0 has fifteen transitions. If each transition accounts for a small section of the transfer function which is inaccurate, this accounts for a much larger portion of the transfer function for less significant bits.

Taking the four bit signals together, they are correct an average of 94 % of the time. However, there's only 81 % of the domain where *all four* bit-signals are correct, since the areas where any of the signals are incorrect don't necessarily overlap.

In a realistic system, it might not be realistic to say that the signal is high whenever it's above the halfway point and low whenever it's below. Instead, let's consider a fairly conservative eye requirement where a subsequent system needs more pronounced value to determine whether the signal is high or low. Let's say that if the output signal of the ADC is above or equal to 150 mV, the signal will definitely be interpreted as a binary 1. Similarly, we'll say that the signal should be below or equal to 50 mV to be a binary 0.

With these constraints in mind, let's look again at the ADC output signals, in Figure 4.13. The same signals are plotted here, except the digital values of these signals are plotted as well. The thresholds of 50 mV and 150 mV are shown as grey lines, and a black line is plotted whenever the signals are below or above these respective thresholds to indicate the binary values of the signals.

We can observe a few things. V_{b_3} is least affected by these stricter constraints, due to the large jump in this signal. This jump almost completely bridges the gap between the two thresholds. As a consequence, this signal still produces the correct digital value 93 % of the time.

For the other signals, which don't have such a jump, their small amplitudes are problematic. These signals only barely reach the thresholds, creating unambiguous digital signals for only a small percentage of the input domain. This is especially true in the case of V_{b_2}, which only produces the right digital value 48 % of the time. It never produces the *wrong* value unambiguously, but it also doesn't reach an unambiguous level in many cases.

This is true to some extent for the two other signals as well. On average, the signals thus produce the correct binary values only 63 % of the time. When combining the signals into a four-bit word, the situation is even worse: this word is only correct for 18 % of the input signal values. After all, if only one bit signal is ambiguous, that's enough for their combined value to be ambiguous as well. So the combined word is only unambiguous, let alone correct, whenever all four signals are unambiguous. As it turns out, this is the case in 81 % of the input signal's values.

Since the small amplitude of the signals, and the consequent ambiguous digital value, pose such a problem, let's see what our amplification circuit can do to help us here.

4.3.5. ADC circuit with amplification

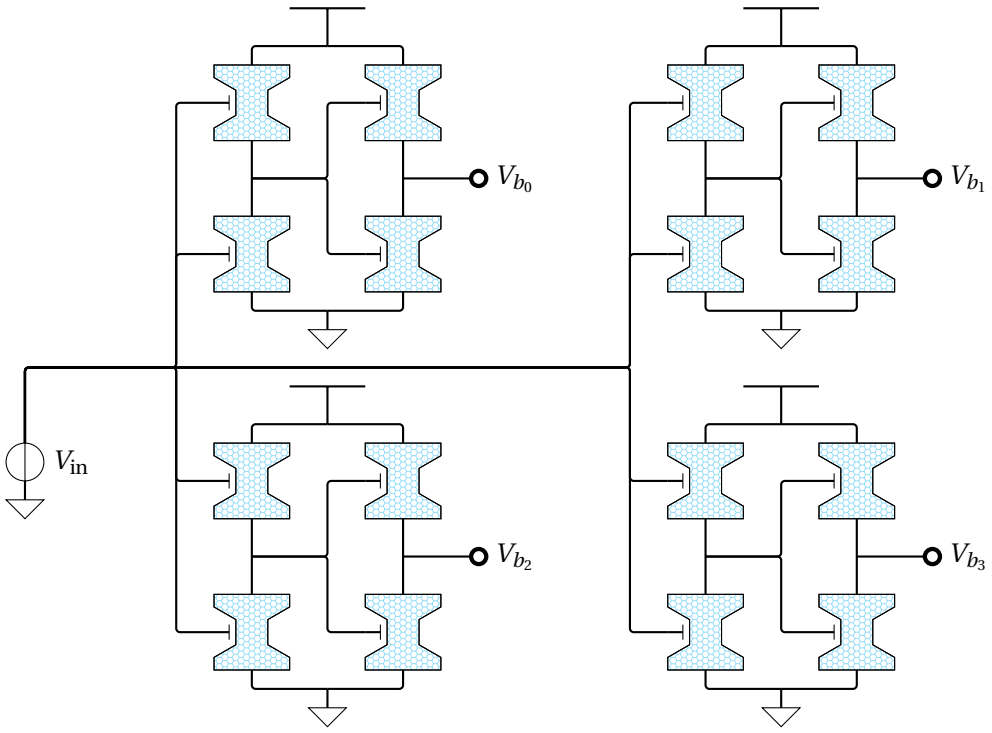
When we add an amplifier to each of the ADC's bit's subcircuits, we get the circuit presented in Figure 4.14a. Again, each of these amplifiers is the same as the simple circuit of bit 3, appended as a second stage after each bit's conversion stage, just like in Section 4.3.3. For the sake of not having an overly complex figure, each GNR component is displayed in a simplified way which doesn't capture the specific GNR shapes.

Figure 4.14b presents the results of a SPICE simulation of this circuit. Again, the—now amplified—output voltage signal of each bit is plotted with respect to the input voltage which is shared among the four bits. In each of the plots, the desired transitions are marked with gray vertical lines, and the desired digital values in each of these segments are marked as black dotted lines. The thresholds of 50 mV and 150 mV, which are assumed to be sufficient for a digital circuit to unambiguously identify the two binary values, are marked with two gray horizontal lines. The resulting interpreted digital values of the signals are marked as black lines.

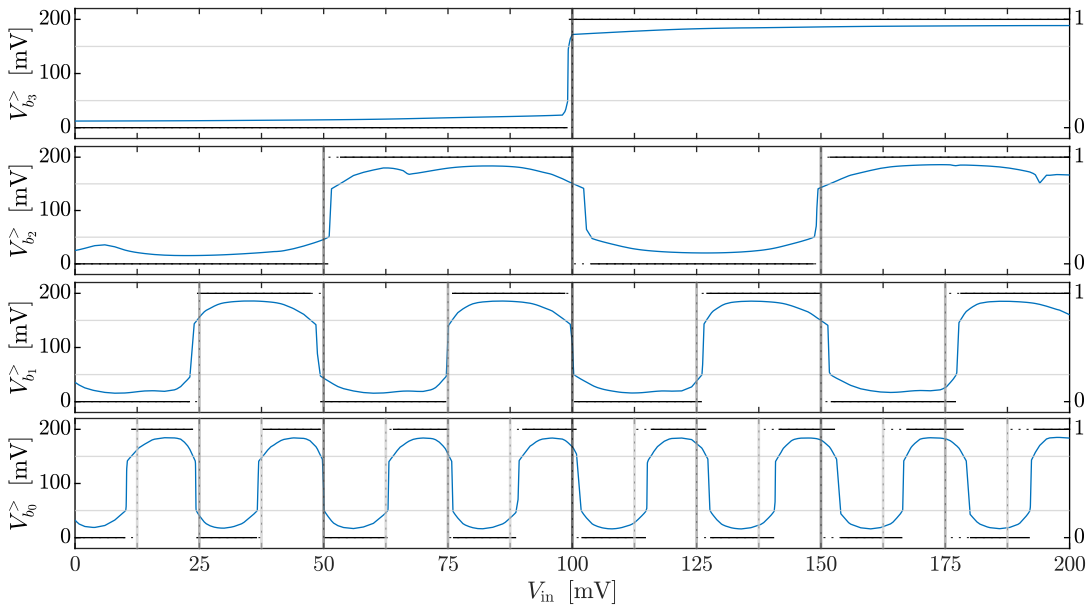
We can observe that the amplification works much as expected: Each signal transitions sharply, almost vertically, between low and high states. As a result, the signal is unambiguous only at transition points. We can observe that the transition positions with respect to the input signal were not affected by the amplification. The points where the unamplified signals crossed the halfway point are the same points where the amplified signals transition between binary states. As such, when the original signal is wrong, so is the amplified signal.

We can observe this in the case of $V_{b_0}^{>}$, the amplified signal for bit 0 of the ADC. Whereas the original, unamplified, signal was ambiguous for a significant portion of input signal values (49 %ofthem), this is reduced to only 7 %. However, the signal now produces the wrong binary value for a sizeable chunk of input signal values. We can observe in the plot that the transitions between binary states drift away from where they should be as V_{in} gets larger. It turns out that this amplified signal produces the desired binary value 80 % of the time, which was 48 % before amplification. The amplified signal produces the wrong binary value in 13 % of cases, up from 2 % before amplification. The signal was ambiguous in those cases before amplification.

On the other hand, $V_{b_3}^{>}$ hasn't changed much. The transition was pretty sharp originally, and it has gotten sharper. The percentage of ambiguous output occurence has decreased from 7 % to 0.1 %,

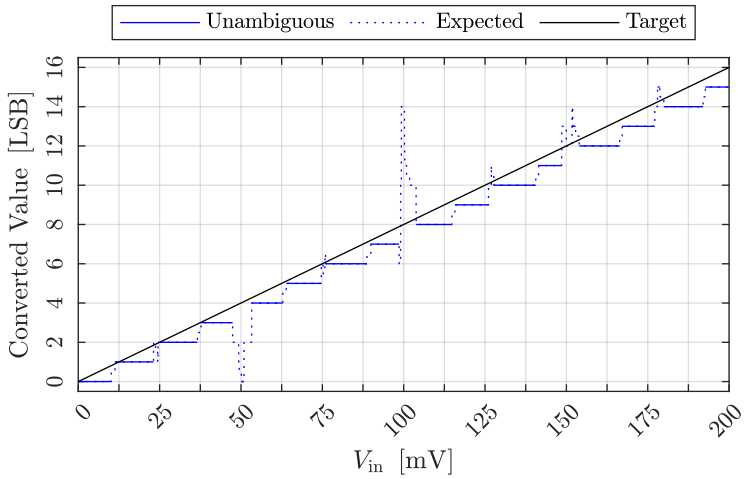


(a) The circuit itself, consisting of four subcircuits for the four bits, each consisting of a conversion stage and an amplification stage.

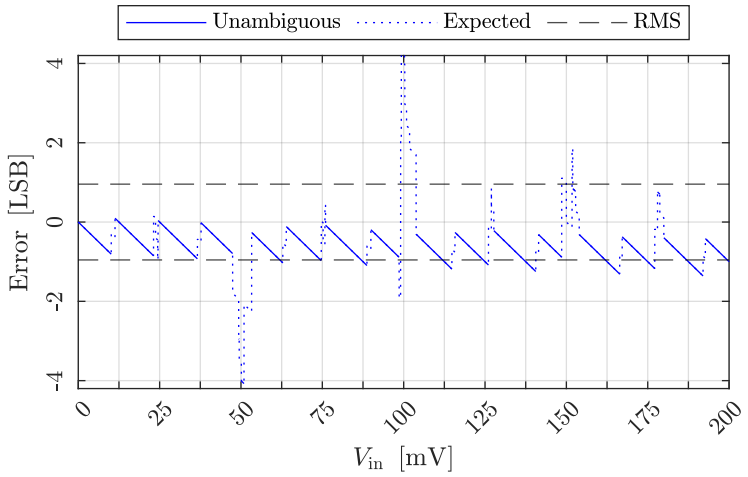


(b) A simulation of this circuit plotting each bit's output voltage signal with respect to the input voltage.

Figure 4.14: The full 4-bit ADC circuit with an amplifier stage for each bit.



(a) The converted total value of the four-bit signal plotted with respect to the input value.



(b) The error of the signal at each input value.

Figure 4.15: The combined value of the output bits of the amplified ADC circuit as compared to the desired target value.

and the percentage where the binary value is correct has increased from 93 % to 99.5 %. Similar results are achieved for the other bits.

Taking the four binary signals together, the ADC output value is unambiguous 85 % of the time and correct 74 % of the time. To be able to relate these numbers to existing ADC designs, it would be nice to have some measure of the output error rate so as to be able to express the accuracy of the system in terms of SNR. To do this, it is important to define what happens when the binary value of the signal is ambiguous. We assume that the subsequent digital system interprets the ambiguous signal randomly as either a 0 or 1 with equal probability. We can then define an expected value for the four-bit combined output value for each input value.

Figure 4.15a presents the combined output value of the ADC circuit with an amplification stage, plotted with respect to its input value. The scale of this plot is in LSB units. The line is solid whenever all four bits are unambiguous and dotted wherever any of the bits is ambiguous. In those situations, what is plotted is the expected value of the combined value of the output bits. In black, the analog input value is plotted, scaled to the same range. This is the target value, meaning the converted digital signal should ideally be as close to this as possible.

Number of amplifier stages	RMS Error [LSB]	SNR [dB]	Effective resolution [bit]
0	1.521	11.40	2.60
1	*0.956*	*15.44*	*3.28*
2	1.086	14.34	3.09
3	1.091	14.29	3.08
4	1.092	14.29	3.08

Table 4.1: The metrics pertaining to noise of the ADC circuit for various numbers of amplifier stages.

In Figure 4.15b, we provide the error of the ADC signal with respect to the target value. The error here being the same as the difference between the actual signal value and the target value. Again, the plotted signal is solid where it's unambiguous and dotted where it's an expected value. In dashed lines, the positive and negative RMS value of this error signal is indicated. We can see quite tall, but narrow, peaks in error around the transition points of the more significant bits, which falls within the line of expectations given that the bit signals don't quite transition at the right points.

The RMS value of this error amounts to 0.96 LSB. In terms of the input signal's range, the RMS equals 12 mV. From this RMS value, we can calculate a SNR value, which turns out to be a ratio of 5.9, or 15 dB. Using the modified formula for an effective number of bits discussed in Section 2.3, we can calculate the effective resolution of the amplified ADC circuit to be 3.3 bits.

The error is affected by the number of amplifier stages, and therefore so are the resultant metrics of RMS noise, SNR, and effective resolution. Table 4.1 presents these metrics for the ADC circuit, augmented by different numbers of amplifier stages. The situation with one amplifier stage, which is what we've been discussing, is *emphasized* in the table.

We can observe that the amplification stage significantly helps reduce noise by removing ambiguity in the signal, amounting to about `0.7 bits` of depth. However, this can only help so much. Where the signals produced by the ADC circuit are misaligned in their transition points, no amount of amplification is going to help. Due to this, more amplification stages don't actually reduce the circuit error. In fact, each amplifier stage beyond the first actually slightly *reduces* the accuracy of the circuit, simply by amplifying the inherent error of the circuit.

The accuracy of the circuit isn't the only thing we care about, of course. There are some other metrics that are important to be able to compare the circuit to the state of the art.

4.3.6. Metrics and the state of the art

By the methods discussed in earlier chapters, we can calculate the relevant circuit metrics. To do that, we first determine the component parameters of each GNR device's equivalent circuit: the gate capacitance C_g, the contact resistance R_C, and the GNR channel resistance R_{GNR}. The gate capacitance and contact resistance are calculated from device dimensions. For the channel resistance we take the worst-case situation, or the reciprocal of the lowest conduction value in the conduction map.

From these component parameters, we can calculate for each GNR device the propagation delay, as well as the power consumption. These can then be combined into a total transition energy. Another relevant measure is the active area of each GNR component, which is calculated by assuming a rectangle of the same outer dimensions as the GNR. That is, a rectangle with the same length as the total length of the GNR and the same height as the total height of the GNR.

Since each bit of the ADC circuit consists of a complementary pair of GNR devices, it is also useful

		Propagation delay (τ_p) [as]	Active area (A) [(nm)^2]	Power (P) [nW]	Transition energy (E_t) [yJ]
GNR ADC					
	Bit 3	4.433	37.40	229.1	1015
	Bit 2	3.029	39.78	173.5	526
	Bit 1	0.648	60.87	215.4	140
	Bit 0	3.729	65.27	179.6	670
GNR Boolean functions [32]					
	AND	27.90	24.28	41.35	1156
	NAND	19.31	27.19	72.18	1394
	OR	12.71	24.53	35.05	445.3
	NOR	19.48	26.98	98.84	1925
	XOR	10.86	24.28	81.65	886.4
	XNOR	16.02	24.52	60.05	961.7

Table 4.2: A comparison in terms of different metrics between the ADC's subcircuits and Jiang et al. [32]'s GNR implementations of 2-input Boolean gates.

to look at these metrics for the combination of two GNR devices. In the case of active area and power consumption, this amounts to simply the sum of the metrics for the PUN and PDN. For the delay, we take the worst case of the two components, assuming parallel propagation through the two. For the transition energy, then, we take the product of the combined delay and total power consumption.

Table 4.2 presents these metrics for each bit's subcircuit. Note that the uncommonly used SI-prefix "y" is used to denote the "yoctojoule", or 10^{-24} J, for the transition energy E_t. The table also includes the metric values for the complementary GNR implementation of 2-input logic gates from Jiang et al. [32].

Comparing the numbers, we can observe that the differences in terms of these metrics between the ADC subcircuits and the Boolean gates are significant, but the values are comparable. The active area A is generally a few times larger in the case of the ADC, while the power P is a few times larger. The propagation delay τ_p exhibits a larger difference, where in the most extreme case the bit 1 subcircuit is 43 times faster than the AND-gate. Combining the power and delay gives a transition energy E_t for the ADC on a very comparable scale to that of the Boolean circuits. In their work, Jiang et al. compares these Boolean gates to similar circuits in CMOS, showing an improvement of several orders of magnitude on all fronts.

This comparison suggests that our numbers are at least in the same ballpark as those of previous work. We didn't expect the ADC bit circuits to score much different from Jiang et al.'s Boolean gates, so this is reassuring.

To get more insight in the potential of our proposal, we present in Table 4.3 an extended comparison with state-or-the-art ADC counterparts. The unamplified, amplified, and doubly-amplified versions of our GNR ADC circuit are compared to various CMOS ADC designs. These designs are compared in terms of several different metrics: The effective resolution of the converter (*ENOB*) when taking noise and error into account, the power consumption of the design (P), the converter's sample frequency (F_s), which are combined into the Walden figures of merit of the time and energy efficiency (P_{Walden}, F_{Walden}), and finally the effective circuit area (A).

ADC Type	Effective resolution $R_{\text{eff.}}$ [bit]	Power P[mW]	Sample frequency $F_{\text{s}}\left[\frac{\text{Gsample}}{\text{s}}\right]$
GNR	2.60	$0.80 \cdot 10^{-3}$	$226 \cdot 10^{3}$
Amplified GNR	3.27	$1.71 \cdot 10^{-3}$	$113 \cdot 10^{3}$
Doubly-amplified GNR	3.09	$2.63 \cdot 10^{-3}$	$75.2 \cdot 10^{3}$
32 nm 6 bit TI flash [9]	4.81	70	20
65 nm 6 bit TI MBS [7]	4.62	88	25
16 nm 8 bit TI SAR [18]	4.9	280	28
28 nm 6 bit TI SAR-TDC [65]	4.51	23	24
7 nm 8 bit TI SAR [49]	4.6	150	28
16 nm 8 bit TI flash-TDC [34]	5.6	175	20
65 nm 8 bit TI TD [68]	6.15	130	20
40 nm 6 bit TI 2S flash [45]	4.71	56	20

ADC Type	Time efficiency $P_{\text{Walden}}\left[\frac{\text{Tstep}}{\text{s}}\right]$	Energy efficiency $F_{\text{Walden}}\left[\frac{\text{Tstep}}{\text{J}}\right]$	Effective area $A\left[(\text{mm})^2\right]$
GNR	$1.37 \cdot 10^{3}$	$1.72 \cdot 10^{9}$	$0.20 \cdot 10^{-9}$
Amplified GNR	$1.09 \cdot 10^{3}$	$0.64 \cdot 10^{9}$	$0.35 \cdot 10^{-9}$
Doubly-amplified GNR	$0.64 \cdot 10^{3}$	$0.24 \cdot 10^{9}$	$0.50 \cdot 10^{-9}$
32 nm 6 bit TI flash [9]	0.56	8.1	0.25
65 nm 6 bit TI MBS [7]	0.62	7.0	0.24
16 nm 8 bit TI SAR [18]	0.84	3.0	2.8
28 nm 6 bit TI SAR-TDC [65]	0.55	23.8	0.03
7 nm 8 bit TI SAR [49]	0.70	4.7	0.09
16 nm 8 bit TI flash-TDC [34]	0.97	5.5	0.1
65 nm 8 bit TI TD [68]	1.42	10.9	0.22
40 nm 6 bit TI 2S flash [45]	0.52	9.3	0.1

Table 4.3: Metrics of the GNR ADC as compared to other ADCs.

Two things stand out immediately. One is that the amplified ADC circuit doesn't actually score better in either of the combined figures of merit than the unamplified circuit, despite having a better effective resolution. This makes logical sense. While these figures of merit grow exponentially with respect to resolution, delay and power usage are roughly doubled by adding an amplification stage. So if this amplification doesn't add a full bit of effective resolution, it won't be worth it when judged by these figures of merit. In some contexts, however, the added resolution might be more important than the time or energy cost of conversion.

The second thing that's noteworthy is the sheer difference in how the GNR circuit scores when compared to conventional ADC circuits. The GNR implementation scores at least several hundred times better in each category other than resolution, scoring roughly a hundred million times better in energy efficiency and a billion times better in active area. Especially considering this, an amplification stage might still be worth the extra power and latency cost.

It makes sense that our design would perform so much better than conventional CMOS ADC designs. After all, these designs consist of many transistor devices making up the different functional stages. Each of those transistors adds functional area, as well as capacitances that need to be charged and discharged, all at the cost of power and time. Each of those functional stages also add more overhead and delay, not to mention clocking. Meanwhile, our base design consists of just eight GNR devices in one functional stage, with the amplification stages adding eight more devices each.

However, there is also some unfairness in this comparison. Our design only takes into account the GNR devices in terms of area and power cost, with no interconnect or overhead, while the CMOS designs in the literature are complete functional ICs in the real world. In addition to this, the delay and power models in this work are very simple, which likely don't take many real-world effects into account, which do play a part in the measurements performed on these CMOS designs. We also don't assume any signal noise in addition to our design mismatch, which obviously wouldn't be valid in a realistic scenario. This would further reduce the effective resolution of a real implementation of our design.

On top of all that, the comparison is skewed by the fact that each of the CMOS designs has a higher nominal resolution than our four-bit ADC. This is an issue because conventional ADC designs often scale very steeply in complexity, area, delay, and power, with increasing resolution. For many practical situations, an effective resolution of around three bits might also simply be too low, so a more useful version of this ADC design would have a higher resolution. In the next section we will look at the implications of extending this design to accommodate more bits.

4.4. ADCs with Larger Resolution

First of all let's look at the performance of the bits we have on an individual basis. Table 4.4 presents the noise and effective resolution of the ADC if we have fewer than four bits. So if we only take a

number of bits	SNR	SNR [dB]	Full accuracy	Effective resolution [bit]
1	1.23	1.82	99.53%	1.00
2	2.41	7.66	94.39%	1.98
3	4.26	12.59	88.81%	2.79
4	5.91	15.44	74.18%	3.27

Table 4.4: Error metrics of the ADC circuit when disregarding one or more bits of the circuit. Each of these is with one stage of amplification. "Full accuracy" refers to the percentage of the input range for which each bit of the ADC produces the correct binary value.

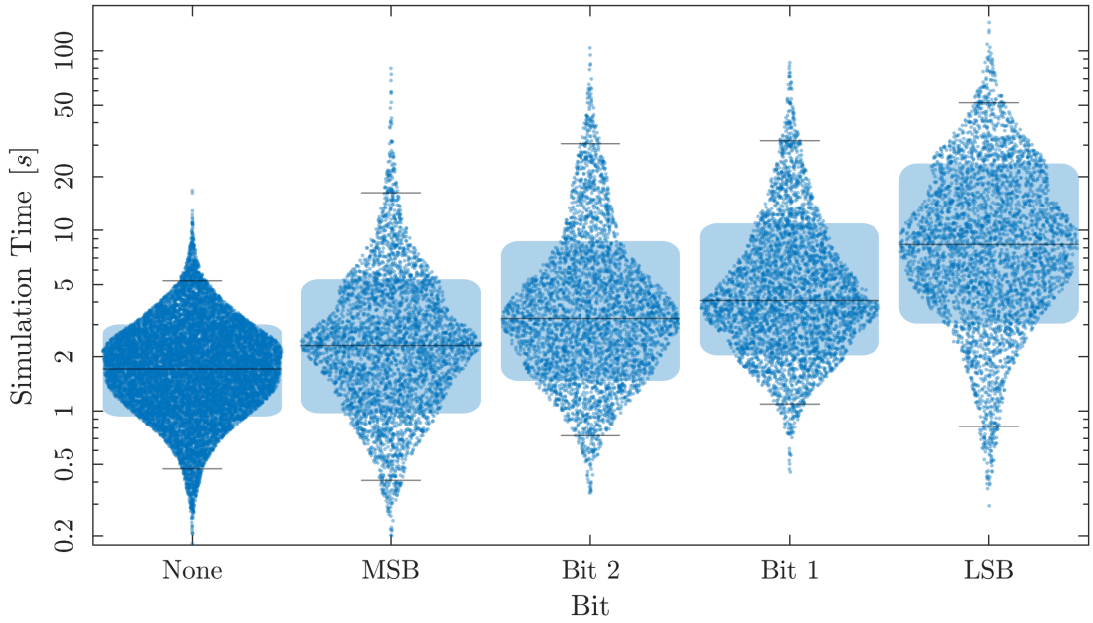


Figure 4.16: The distribution of simulation times for devices of each bit of the ADC circuit.

smaller set of bit subcircuits to make an ADC with lower nominal resolution. From what we can see, there's diminishing returns. Each successive added bit of nominal resolution adds less to the effective resolution. This makes sense, since we've noticed earlier that the less significant bits with more transitions in their transfer function introduce more errors due to misalignment. This is likely caused by a combination of two things. First, the evolutionary algorithm has been unable to find higher fitness devices according to the defined metrics, either because such devices don't exist or because the search isn't extensive enough, as discussed in Section 4.1. Secondly, bits of lower significance simply have more transitions, so there's more potential for misalignment, creating a larger effect for the same amount of relative error. In other words, each additional bit is more difficult to get right than the previous. The implication of this is that it might be difficult to keep extending the system to more and more bits.

Potentially, a more extensive search could result in better GNR devices for additional bits, but this would then require more and more processing time to perform the search, as more devices would need to be evaluated. Speaking of processing time, let's take a look at what is actually taking up the majority of time in the evolutionary algorithm: simulating each device's behavior.

Figure 4.16 reports the time required to simulate the behavior of GNR devices for each bit. Each dot on the figure represents one device that is simulated once. On the x axis, they are separated by category. The "None" devices inhabit the device pools of random devices at the start of each epoch, before any evolutionary selection has taken place. The devices in each of the bit categories then inhabit the device pools are the end of each epoch, after selection each of the four bits of the ADC. These devices are scattered such that their y-position corresponds to the time required to simulate their behavior. The y-axis is logarithmic. They are then spread out in the x direction according to their density, to create a histogram effect. As such, the bulges in the shape indicate that many devices took approximately the same amount of time to simulate.

We can see that the simulation time for the unselected devices in the "none" category roughly follows

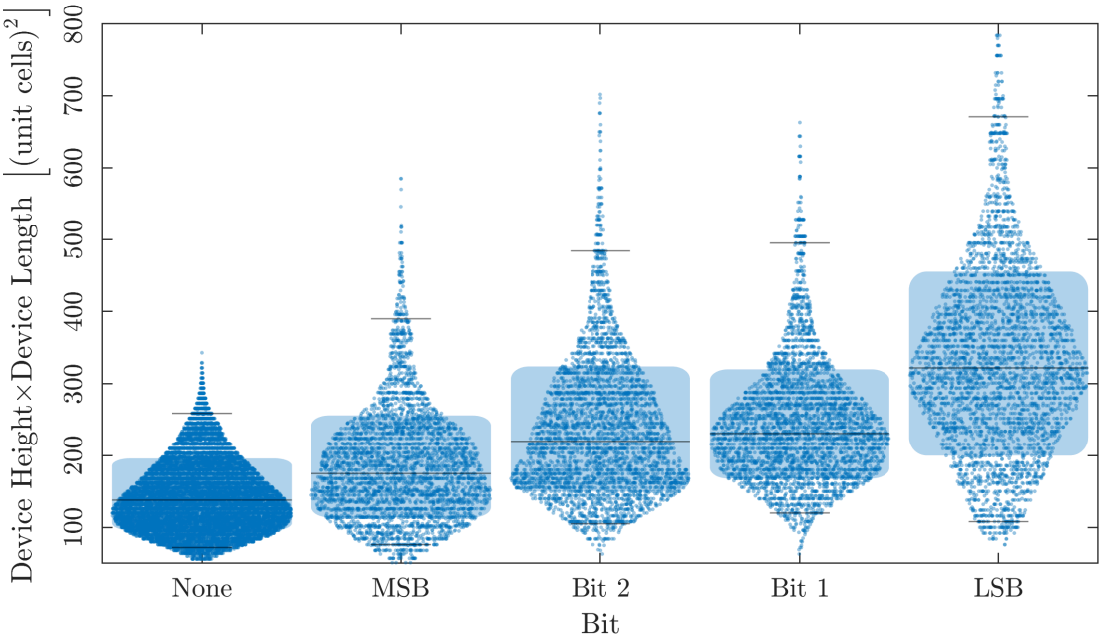


Figure 4.17: The distribution of device size for devices of each bit of the ADC circuit.

a lognormal distribution, with a geometric mean μ of approximately $10^{0.2} \approx 1.7$ s and geometric standard deviation σ of approximately $10^{0.3} \approx 1.8$. This means that around 68 % of pre-evolution devices are simulated in more than $\mu/\sigma \approx 0.9$ s but less than $\mu \cdot \sigma \approx 3.0$ s. In other words, the 16^{th} percentile of simulation times is close to μ/σ, and the 84^{th} percentile to $\mu \cdot \sigma$. Similarly, the 2^{nd} and 98^{th} percentiles are respectively μ/σ^2 and $\mu \cdot \sigma^2$, so about 95 % of devices are simulated within that interval. These percentiles are marked: a blue box marks the range between the 16^{th} and 84^{th} percentile and black lines mark the 2^{nd}, and 98^{th} percentiles, as well as the 50^{th} percentile (or the median). While the evolved devices for each bit don't follow the lognormal distribution as nicely, the same percentiles are marked.

We can plainly see that devices selected through evolution for each less significant bit require a longer time for behavioral simulation. The median device for bit 1 takes 1.8 times as long to be simulated compared to the median MSB device. This is even worse for the median LSB device, which takes 3.6 times as long as the median MSB device.

To look for an explanation of this trend, we should consider the size of devices, plotted in Figure 4.17. The size of these devices is expressed in the figure as the product of their length and height, both of which contribute to larger matrix operations to be done in the simulation framework. We can observe that lower significance bits are larger than higher significance bits, which explains why they need more time to be simulated.

If this trend continues, adding more bits to the ADC circuit will require increasingly more time to simulate the behavior of those bits' devices. When combined with the trend that the evolutionary algorithm seems to have a tougher time finding suitable device topologies for less significant bits, further bits will require ever more computing. Adding on to this, larger devices mean a larger solution space to be searched.

This is not to say a five or six bit ADC wouldn't be possible. There's no reason to assume the evolution-

ary algorithm couldn't be made more efficient or clever, or that more computing resources couldn't be utilized. But sooner or later, depending on how strongly these trends continue, these problems might grow to be prohibitive.

Apart from the difficulties in finding suitable GNR device topologies to implement further bits, there is also the problem of transitions, which form the main source of error. At each point in the transfer function of a bit where the desired bit value transitions between `0` and `1` there's the possibility of misalignment between the PUN and PDN conduction maps. For each additional bit, the number of these transitions doubles, introducing more potential error. If transition inaccuracy isn't managed, transition noise will dominate quantization noise eventually, making additional bits useless.

More conventional methods might also be utilized in extending the resolution of the system. Given a suitable digital-to-analog converter, and analog subtraction circuit, a successive approximation setup could be possible, where each approximation step utilizes a GNR ADC circuit with few bits.

Some of the error of the system could also be mitigated by having multiple ADC circuits and a majority vote. If each ADC circuit has slightly different transition points in its transfer functions in such a way that some slightly overshoot the desired transition and some fall slightly short, this could average out and the combined system would be more accurate.

Another solution may lie in the adding of redundant bits. Rather than having each subcircuit encode exactly one bit of information about the input signal, a scheme could be devised where the subcircuits each produce overlapping information, such that a more robust reconstruction of the input signal could be determined from them.

But all of that is for another time.

5

Conclusion

In this thesis, we attempt to design an evolutionary algorithm which could be utilized to efficiently find a GNR device geometry which exhibits some arbitrary behavior. We then use utilize algorithm to find a set of GNR devices which, when put together, perform the functionality of a four-bit ADC circuit.

To do so, we observe the characteristics of the transfer functions of an ADC circuit. From there we conclude that if we design GNR devices with specific conduction maps, we can utilize them to design a new type of ADC circuit. We then build an evolutionary algorithm which can search an arbitrarily large space of possible GNR devices for suitable devices according to some specified fitness function. Finally we design such a fitness function which combines all the characteristics we want from our GNR devices so the evolutionary algorithm can search for all the devices we need to create our ADC circuit.

After running this evolutionary algorithm, we first consider the process of it. We find that the evolutionary algorithm has found devices with increasing fitness over time for each of the different target devices. After observing the self-similarity of the algorithm's device pool over time, we conclude that in most runs, the algorithm converges to some local optimum within fifteen generations, and that these local optima have been reached repeatedly in multiple epochs, but that different distinct local optima have been reached across the epochs.

When considering the devices produced by the algorithm on their own, we note that different epochs have indeed produced distinct classes of similar devices, corresponding to these distinct local optima. We also in fact observe that the found devices correspond quite well to the requirements we set and the behavior we desired. Taking into consideration specifically the found devices for one certain target device required for the circuit, we discuss the difficulties of picking the right device from multiple mostly-suitable candidate devices. In the end we conclude that hand-picking from the top candidates was most practical at that point.

Finally we consider the ADC circuit using these GNR devices and how it compares to conventional ADC circuits. Putting the PUN and PDN GNR devices together into complementary circuits, we find that the resulting transfer functions corresponds quite well to the behavior we want for our ADC circuit. While investigating the behavior of the circuit corresponding to the MSB we also conclude that this circuit could handily be reused to amplify and disambiguate our imperfect output signals into clear digital values.

Putting all the bits together we judge the total circuit and its performance as an ADC. In terms of

accuracy, our circuit performs as well as an idealized ADC circuit with a bit depth of 3.3 bits.

Where the ADC circuit really shines, though, is in its efficiency. When compared to other complementary GNR circuits the cost of our ADC in terms of propagation delay, circuit area, power cost and transition energy is of comparable magnitude to elementary boolean operations, whereas conventional CMOS ADCs cost many times more than these boolean operations. We observe, when we comparing our ADC circuit to some conventional ADC designs, that our GNR design is much more efficient, even though it didn't have as high a resolution: It consumes tens of thousands of times less power;it has a sample frequency several thousands times higher;as a result it has a Walden time efficiency of around a thousand times higher and Walden energy efficiency of nearly a billion times higher; and all that in an effective circuit area of about a billion times less.

Finally we observe at some trends that hint at the implications of extending our circuit to a higher resolution. We find that more bits of nominal resolution would likely have diminishing returns in terms of effective resolution. In addition to this, we presume that it would likely be increasingly hard to find suitable GNR devices to increase the ADC resolution, noting that the device fitness at convergence was not as good for the higher bits, and the increasing number of transitions would make alignment of conduction maps more difficult. We also look at some trends in the cost of our evolutionary algorithm, noticing that bits of lower significance (which are required for a higher resolution ADC) require larger GNR devices, which in turn take a longer time to simulate the behavior of. In all, we conclude that it likely wouldn't be impossible to extend our ADC design to a higher resolution, but it would be increasingly expensive to do so in terms of time to find the correct GNR geometries, and each additional bit would add a decreasing amount of effective resolution as well.

5.1. Research questions

Let's revisit our original research questions and determine what we've found.

- **Can an evolutionary algorithm find GNR geometries to implement some arbitrary functionality?**

In this work we've successfully designed an evolutionary algorithm which finds suitable GNR geometries for eight different functionalities. There is no reason to assume this same algorithm could not be used, with a different fitness function, to find GNR devices with some arbitrary other behavior for whatever purpose.

- **Would such an algorithm be significantly more efficient at searching the solution space than conventional exhaustive methods?**

In terms of efficiency, our algorithm has only evaluated tens of thousands of different GNR devices out of a solution space of many orders of magnitude larger, so that's fairly efficient. Of course there's no conclusive evidence that the best possible GNR device candidates within the solution space have actually been found, which an exhaustive search by definition would. On the other hand, different epochs of the search which started from random initial device pools has kept converging on the same few spaces within the space, which points to those potentially being the only solutions in the space.

- **Can a circuit consisting of only a few GNR devices function as an effective ADC?**

We've successfully designed a functional four-bit ADC circuit using only complementary GNR device circuits. Now the question remains whether this design could be expanded to a more useful resolution, and it seems like there'd be some issues to doing that.

- **Would such an ADC circuit perform significantly better than state-of-the-art circuits in terms of area, energy, and/or time efficiency?**

The ADC circuit using complementary GNR circuits is remarkably efficient. Its accuracy is not perfect, but the operating costs are incredibly low when compared to the state of the art. Specifically, the GNR ADC performs several orders of magnitude better than conventional designs in terms of power, delay, and area cost.

5.2. Future Work

There are a few different general approaches to improving the work presented in this thesis. We'll divide these into ways of improving the evolutionary algorithm to more efficiently find suitable devices for some functionality on the one hand, and ways of improving the ADC circuit design to perform better on the other hand.

Let's start with some ways of improving the evolutionary algorithm. One problem with the algorithm as it stands, is that it isn't able to take one device of a complementary pair into account when looking for the other. That is, when searching for the PUN device, it would be best if we knew exactly the behavior of the PDN device so we can make them match up, and vice versa. In more technical terms, the fitness function used to search for one device could be made dependent on the behavior of the corresponding complementary device. The problem lies in this fact that it applies to both devices of the pair, though. It'd be possible to wait for the search for the PUN to be done to inform the search for the PDN, or the other way around, but you can't do both.

Instead, it might be possible to solve this problem some other way. One way is to treat a pair of GNR devices together as one organism in the evolutionary algorithm, evaluating their transfer function in the fitness function. This would allow for more direct selection of the behavior you want from the *circuit*, rather than selecting for the behavior of the individual *components* and hoping they match. This would come at the cost of having much more complex organisms with more parameters.

Another way to deal with this issue would be to take advantage of the parallelizability of the evolutionary algorithm. If we are performing two independent searches in parallel for both GNR devices in the complementary pair, then some way could be devised to make the two searches influence each other. For example, the current best-fitting PUN device could be used to periodically update the fitness function used in the search for the PDN device, and vice versa. One thing to take into account here is that a circular dependency is created, so care should be taken to make sure this is stable.

There are other, simpler ways to improve the algorithm. The size of the search space can be increased by allowing more or larger GNR shapes. The algorithm can be made to halt after reaching some measure of convergence so as to always have an optimal number of generations to not waste computational time. The process of creating random new devices for a new epoch can be biased away from already searched portions of the search space to improve the chances of finding new local optima. More thought can be put into tuning the weights of the fitness function. Or of course more computational resources can be utilized to just run the algorithm for more generations.

Whether the evolutionary algorithm is further improved or not, it could also be used to search for GNR devices for other purposes than ADC circuits. As mentioned before, GNRs have previously been used to implement artificial neurons and Boolean gates previously [28, 59]. But there are bound to be more potential applications for GNR circuits with arbitrary transfer functions, for example performing analog functions. Whichever use for GNR circuits is thought of, this evolutionary algorithm could be used to find the required devices.

Independently from any efforts to improve the evolutionary algorithm, there are a few ways to

improve on our GNR ADC circuit design as well. One of the problems of our circuit lies in the fact that the transition points of the different transfer functions lie at the same point in the input signal's range. This overlap of the error-prone segments of the transfer functions allow for the possibility of simultaneous errors, which are more problematic than single-bit errors. This overlap of transition points is ultimately caused by the way we aim to encode our digital symbol values: with a positional numeral system. Such a system is simple, reminiscent of familiar everyday decimal notation, and useful for mathematical operations, but it's not perfect for any circumstance. If we instead use some other arrangement of our different bit values with respect to the input signal, we might prevent some problems.

For example, the bit circuits may be designed such that they follow the system of so-called Gray codes instead [21]. Such a system rearranges the different binary symbols to only require a single bit to transition at a time when traversing the signal range gradually.

Alternatively, additional bit circuits may be introduced which don't add further resolution, but rather add redundancy to the digital output. These redundant bits might be designed such that when one or more of the conventional bit circuits would struggle to produce the correct output, the redundancy bits could provide clarity.

Such schemes which deviate from the standard positional numeral system would add complexity, though. A stage of digital logic circuitry would be required to convert this scheme back to standard binary notation to allow for further digital arithmetic. This added cost might be worth the effort if it means the resolution or accuracy of the ADC could be improved, however.

In addition to these further avenues of research pertaining to our unconventional ADC circuit design, more familiar paths might be interesting as well. Our GNR ADC circuit could be used as part of a more conventional ADC technique such as successive approximation: By repeatedly using our 4-bit ADC while biasing and scaling the input signal according to the digital output, more bits of information may be gleaned from the input. Potentially, other conventional ADC design techniques may also be applied to this type of ADC circuit design.

Glossary

2S Two-Step: A class of *analog-to-digital converter* (ADC) circuit consisting of two steps. In the first step, the input signal is converted to a digital symbol using an ADC of some resolution. This digital signal is then converted back to an analog signal through a digital-to-analog converter. This reconstructed signal is then compared with the original signal to determine the (small) difference between the input signal and the converted value of it. This difference (or "residue") signal is then amplified and sent through another ADC circuit in the second step, to determine this residue signal to some resolution. In this way, two low-resolution ADCs can be used to attain a similar precision as a double-resolution ADC.

accuracy The accuracy of one bit subcircuit of the ADC discussed in this work refers to a metric of the *transfer function* of that subcircuit. The accuracy measures the fraction of input values V_{in} which result in the correct digital output value b_n. That is, an output value which matches the desired digital output.

ADC Analog-to-Digital Converter: An electronic circuit which has the function of converting an analog input voltage to one or more bits of digital information symbolizing the value of this input voltage.

ambiguity The ambiguity of one bit subcircuit of the ADC discussed in this work refers to a metric of the transfer function of that subcircuit. The ambiguity measures the fraction of input values V_{in} which result in an ambiguous digital output value b_n. That is, an output signal which can't clearly be determined to be either digital value `0` or `1`.

CMOS Complementary MOS (FET): A family of circuits made of *MOSFET* devices, consisting of a *pull-up network* (PUN) made up of pMOS *transistors* which allow the output signal to be pulled up to a high voltage, and a *pull-down network* (PDN) made up of nMOS transistors which allow the output signal to be pulled down to a low voltage. These two networks are complementary, such that one and only one of the two is active under any circumstance.

conduction map A conduction map is a mapping for the conductance of an electrical component under different circumstances, such as different contact voltages. In this work, this will always refer to a mapping of a *graphene nanoribbon* (GNR) device's gate voltage to its drain-to-source conductance.

DFT Discrete Fourier Transform: A discrete version of the *Fourier transform* which converts a finite number of samples of a signal into the same number of samples of the signal's spectrum.

direct conversion Direct conversion ADCs form a class of ADC circuits which use a series of comparators to determine the level of the input signal. The outputs of these comparators are then combined into a multiple-bit positional binary digital symbol. Named so because the input signal is directly converted to a digital symbol. *See also: flash.*

effective resolution The effective resolution ($R_{\text{eff.}}$) of an ADC circuit is a measure of the precision of that circuit. When combining all relevant error sources that apply to the circuit, a total *signal-to-noise ratio* (SNR) can be determined. The effective resolution is the resolution of an ideal ADC circuit which only exhibits *quantization* noise, the *signal-to-noise ratio* (SNR) of

which is equal to the ADC circuit in question. This can be calculated using the formula:

$$R_{\text{eff.}} = \frac{\text{SNR}_{\text{dB}}}{6.02}$$

, where SNR_{dB} denotes the circuits SNR in decibel. . *See also: resolution.*

epoch An epoch is a certain number of *generations*, after which an *evolutionary algorithm* starts over with a brand new *population* to explore new areas of the *fitness landscape*.

evolutionary algorithm An evolutionary algorithm is any of a family of algorithms which mimic the natural process of evolution to automatically reach a solution to some problem.

fitness In the context of evolution, the fitness of some organism describes how well it is adapted to its environment and its biological imperative of reproduction. A more fit individual is better able to survive, reproduce, and pass its genetic information on to the next generation.

Similarly, in the context of evolutionary algorithms, this is some measure of how good a particular individual candidate solution is at solving the posed problem.

See also: fitness function , fitness landscape .

fitness landscape The fitness landscape, or *fitness surface*, is a surface in a multidimensional space defined by the fitness function, resulting from the mapping of multidimensional individuals into one-dimensional fitness scores. *See also: fitness , fitness function , fitness surface .*

fitness function The fitness function is a mathematical function which takes an individual in some evolutionary algorithm as input and produces a number describing that individual's fitness as output. *See also: fitness , fitness landscape , fitness surface .*

flash Flash ADCs form a class of ADC circuits which use a series of comparators to determine the level of the input signal. The outputs of these comparators are then combined into a multiple-bit positional binary digital symbol. Named so because flash ADCs are generally operate very fast, with a low *propagation delay*. *See also: direct conversion.*

generation A generation, in the context of evolution, refers to all individuals in a population which were born at roughly the same time under the same conditions. Each generation *reproduces* to create the next generation, and is created by the previous generation similarly. A generation can also refer to the amount of time elapsed between the birth of one generation and the next.

GFET Graphene : A *FET* device where the channel is made up of *graphene*.

GNR Graphene Nanoribbon: An electronic device consisting of a specifically shaped patch of graphene with two terminal contacts and optionally one or more gate contact. The acronym "GNR" can refer to either the graphene nanoribbon patch itself, or to the electronic device containing this graphene nanoribbon patch.

graphene Graphene is a molecular structure consisting of a one atom thick sheet of carbon atoms in a hexagonal lattice. *See also Section 2.1.*

latency The latency, or propagation delay (τ_{p}), of an ADC circuit describes the delay between the input signal being sampled and the matching output symbol being produced. *See also: propagation delay , sample frequency , sample rate .*

LSB Least Significant Bit: The lowest-valued bit in the binary representation of a number. Also, a single unit in the value associated with the value associated with that binary number. *See also: MSB.*

MBS Multi-Bit Search

MOSFET Metal–Oxide–Semiconductor Field-Effect Transistor: A type of FET consisting of a metal gate terminal isolated by an oxide dielectric from a semiconductor substrate. These are often made in two variante: p-channel (pMOS), and n-channel (nMOS), with different doping levels in the semiconductor substrate so as to create inverted behavior. As such, the channel of a pMOS transistor conducts when the gate voltage is low, while an nMOS conducts when the gate voltage is high.

MSB Most Significant Bit: The highest-valued bit in the binary representation of a number. *See also: LSB.*

mutation A mutation is a slight change in the offspring of a parent in an evolutionary process, which allows the offspring to perform differently from its parents, potentially improving upon them.

PDN Pull-Down Network: in a complementary logic circuit, the pull-down network is used to set the output to logic `0`. The conductivity of the PDN is high if and only if the logic value of the required logic function is `false`. This PDN then connects the output net to V_{DD}. *See also: PUN.*

PDP Power-Delay Product: The product of the power consumption of a circuit such as an ADC, with the propagation delay of that circuit. The result is a measure of the energy expenditure of a single operation. Often the worst case for both values is taken. *See also: transition energy .*

population In an evolutionary context, a population is a collection of evolving individuals. All individuals in this population are subject to the same selection criteria and reproduction mechanisms.

propagation delay The propagation delay (τ_{p}), or latency, of an ADC circuit describes the delay between the input signal being sampled and the matching output symbol being produced. *See also: latency , sample frequency , sample rate .*

PUN Pull-Up Network: in a complementary logic circuit, the pull-up network is used to set the output to logic `1`. The conductivity of the PUN is high if and only if the logic value of the required digital function is `true`. This PUN then connects the output net to V_{SS}. *See also: PDN.*

quantile A quantile is a measure of some dataset dividing all its values such that a specified number of these values fall above and below the division. For example, you can divide a dataset into four quartiles by making three quantile divisions, where the highest 25 percent of values fall into the highest quartile above the 25 % quantile, and the next quarter of all data points fall between the 25 % and 50 % quantiles, etcetera. Similarly, you can make a quantile division for any real number p between 0 and 1 of a dataset of N points, such that the highest $p \cdot N$ points have values above that quantile value, and the other $(1-p) \cdot N$ points are values below it.

quantization The inaccuracy of an ADC circuit based on its limited resolution. *See also: resolution , resolution .*

reproduction In an evolutionary context, reproduction refers to the process where an individual from one generation produces offspring which is similar (but not identical, due to mutation) to itself in order to introduce new individuals to the population of the next generation.

resolution The level of precision with which an ADC circuit can capture the value of its input signal, expressed as a number of bits. For example, a `4`-bit ADC circuit distinguishes between $2^4 = 16$ different levels in the input signal, and converts these to a 4-bit digital symbol.

In real-world implementations of ADC circuits, resolution is limited by noise. This gives rise to the phrase effective resolution, which is a measure of how much information the circuit can

measure after taking the noise into account. *See also: effective resolution , quantization .*

RMS Root Mean Square: x A measure of the mean value of a signal. Defined as

$$\mathrm{RMS}\{s(x)\} = \sqrt{\mathrm{mean}\{s(x)^2\}}.$$

sample rate Sample rate, or sample frequency, (F_s) is a measure of the speed of an ADC circuit. It expresses how many times in a second the circuit can sample the input signal and convert it to a digital symbol. A related measure is the propagation delay, or latency. In a simplistic ADC circuit, the sample rate is limited by this delay. However, many ADC circuits utilize parallelism to effectively sample the input signal more often than the latency would allow. *See also: sample frequency , latency , propagation delay , time-interleaving .*

SAR Successive Approximation Register: A type of ADC circuit where the value of the input signal is successively approximated over several steps. In the first step, the value of the input signal is determined to some resolution. A digital-to-analog converter then produces an analog signal corresponding to this imprecise approcimation of the input signal. The difference between this approximation and the original input signal is then used to determine the value of the input signal to a higher resolution. This repeats for any amount of steps.

selection In an evolutionary context, selection describes the process where a subset of individuals in a population is determined who will reproduce. This selection process is according to the fitness of each individual in the population. This fitness might also include a random component.

SNR Signal-to-Noise Ratio: A measure of a signal's strength as compared to the noise associated with that signal. Defined as the ratio of the signal's power to the total noise power. This is equivalent to the square of the ratio of the RMS voltages of the signal and noise. This ratio is often expressed in decibels.

$$\mathrm{SNR_{dB}} = 10 \cdot \log_{10}(\mathrm{SNR}) = 10 \cdot \log_{10}\left(\frac{P_{\mathrm{signal}}}{P_{\mathrm{noise}}}\right) = 20 \cdot \log_{10}\left(\frac{\mathrm{RMS}\left(V_{\mathrm{signal}}\right)}{\mathrm{RMS}\left(V_{\mathrm{noise}}\right)}\right)$$

See also: RMS.

SPICE Simulation Program with Integrated Circuit Emphasis: A commonly used circuit simulator.

TD Time-Domain: A class of ADC circuits which convert the input signal to a time-domain signal before converting that to a digital value. An example of a time-domain signal is a delayed pulse, where the amount of delay is proportional to the magnitude of the input signal. By means of a fast digital counting circuit, this delay can then be determined to determine the input signal's value. *See also: time-to-digital converter.*

TDC Time-to-Digital Converter: A circuit which converts some time-domain signal to a digital symbol, corresponding to the value of that signal. An example of a time-to-digital converter is a circuit which counts clock cycles until stopped. The number of counted clock cycles that occured before stopping the circuit then functions as the digital output value. *See also: time-domain.*

TI Time Interleaved: Time-interleaving is a technique used in ADC design where multiple identical ADC circuits are implemented in parallel and each samples the signal with a slight delay relative to the rest, so you can have multiple time samples in the time it takes one sample to be performed, increasing the sample rate. *See also: latency , propagation delay , sample rate , sample frequency .*

transfer function The transfer function of a (complementary) GNR circuit refers to a mapping between the input signal of that circuit and its output signal. This output signal can be analog or digital, and this function can take any form so long as it maps each input value to exactly one output value. In the case of a complementary circuit, the transfer function follows the formula

$$V_{\text{out}}(V_{\text{in}}) = \frac{G_{\text{PUN}}(V_{\text{in}})}{G_{\text{PUN}}(V_{\text{in}}) + G_{\text{PDN}}(V_{\text{in}})} \cdot V_{\text{DD}}$$

, where $V_{\text{out}}(V_{\text{in}})$ refers to the output voltage of the circuit as a function of the circuit's input voltage V_{in}, $G_{\text{PUN,PDN}}(V_{\text{in}}$ refers to the conductance of respectively the PUN or PDN device, as a function of V_{in}, and V_{DD} refers to the rail-to-rail supply voltage .

transition point In the transfer function of a GNR circuit, the transition point marks the position in the input range, and therefore the input value, where the device transitions from a low output value to a high output value, or vice versa. That is, at input voltages lower than the transition point, the circuit output is low, and at input voltages higher than the transition point, the device output is high, or vice versa. Note that a single transfer function can have multiple transition points.

Similarly, this concept of transition points can also be applied to the conduction map of a GNR device, where the conductance of the device transitions from a low to a high conductance value or vice versa at some gate voltage.

transition energy The energy cost (E_{t}) associated with a single conversion of a ADC circuit, or a single operation of any other circuit. *See also: Power-Delay Product.*

Walden energy efficiency The Walden energy efficiency, or Walden figure of merit (F_{Walden}) is a combined metric for evaluating the performance of an ADC circuit first defined by and named after Robert H. Walden [57]. The intention of this metric is to be able to compare different types of ADC circuits which can have a different (effective) resolution. The number represents the effective energy cost of conversion per step of precision of that conversion. This figure of merit can be calculated using the formula

$$F_{\text{Walden}} = \frac{2^{R_{\text{eff.}}} \cdot F_{\text{s}}}{P} = \frac{P_{\text{Walden}} P}{}$$

, where $R_{\text{eff.}}$ represents the effective resolution of the ADC circuit, F_{s} represents the sample rate of the ADC, P represents its power consumption, and P_{Walden} represents the *Walden time efficiency*. *See also: Walden time efficiency , Walden figures of merit* .

Walden time efficiency The Walden time efficiency, or Walden figure of merit (P_{Walden}) is a combined metric for evaluating the performance of an ADC circuit first defined by and named after Robert H. Walden [57]. The intention of this metric is to be able to compare different types of ADC circuits which can have a different (effective) resolution. The number represents the effective time cost of conversion per step of precision of that conversion. This figure of merit can be calculated using the formula

$$P_{\text{Walden}} = 2^{R_{\text{eff.}}} \cdot F_{\text{s}}$$

, where $R_{\text{eff.}}$ represents the effective resolution of the ADC circuit, and F_{s} represents the sample rate of the ADC. *See also: Walden energy efficiency , Walden figures of merit* .

Bibliography

[1] Waqas Bin Abbas, Felipe Gomez-Cuba, and Michele Zorzi. Millimeter Wave Receiver Efficiency: A Comprehensive Comparison of Beamforming Schemes With Low Resolution ADCs. *IEEE Transactions on Wireless Communications*, 16(12):8131–8146, December 2017. ISSN 1558-2248. doi: 10.1109/TWC.2017.2757919.

[2] G. Baccarani, M.R. Wordeman, and R.H. Dennard. Generalized scaling theory and its application to a $\frac{1}{4}$ micrometer MOSFET design. *IEEE Transactions on Electron Devices*, 31(4):452–462, April 1984. ISSN 1557-9646. doi: 10.1109/T-ED.1984.21550.

[3] Mark Bohr. A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, 2007. ISSN 1098-4232. doi: 10.1109/N-SSC.2007.4785534.

[4] M.T. Bohr. Interconnect scaling-the real limiter to high performance ULSI. In *Proceedings of International Electron Devices Meeting*, pages 241–244, December 1995. doi: 10.1109/IEDM.1995.499187.

[5] K. I. Bolotin, K. J. Sikes, Z. Jiang, M. Klima, G. Fudenberg, J. Hone, P. Kim, and H. L. Stormer. Ultrahigh electron mobility in suspended graphene. *Solid State Communications*, 146(9):351–355, June 2008. ISSN 0038-1098. doi: 10.1016/j.ssc.2008.02.024.

[6] Christopher J Bullock and Cyrill Bussy. Biocompatibility considerations in the design of graphene biomedical materials. *Advanced Materials Interfaces*, 6(11):1900229, 2019. ISSN 2196-7350.

[7] Shengchang Cai, Ehsan Zhian Tabasy, Ayman Shafik, Shiva Kiran, Sebastian Hoyos, and Samuel Palermo. A 25 GS/s 6b TI Two-Stage Multi-Bit Search ADC With Soft-Decision Selection Algorithm in 65 nm CMOS. *IEEE Journal of Solid-State Circuits*, 52(8):2168–2179, August 2017. ISSN 1558-173X. doi: 10.1109/JSSC.2017.2689033.

[8] P.K. Chatterjee, W.R. Hunter, T.C. Holloway, and Y.T. Lin. The impact of scaling laws on the choice of n-channel or p-channel for MOS VLSI. *IEEE Electron Device Letters*, 1(10):220–223, October 1980. ISSN 1558-0563. doi: 10.1109/EDL.1980.25295.

[9] Vanessa H.-C. Chen and Lawrence Pileggi. A 69.5 mW 20 GS/s 6b Time-Interleaved ADC With Embedded Time-to-Digital Calibration in 32 nm CMOS SOI. *IEEE Journal of Solid-State Circuits*, 49(12):2891–2901, December 2014. ISSN 1558-173X. doi: 10.1109/JSSC.2014.2364043.

[10] Jinseok Choi, Gilwon Lee, Ahmed Alkhateeb, Alan Gatherer, Naofal Al-Dhahir, and Brian L. Evans. Advanced Receiver Architectures for Millimeter-Wave Communications with Low-Resolution ADCs. *IEEE Communications Magazine*, 58(8):42–48, August 2020. ISSN 1558-1896. doi: 10.1109/MCOM.001.2000122.

[11] Charles Darwin. *On the Origin of Species*. Routledge, London, 1859. ISBN 978-0-203-50910-4. doi: 10.4324/9780203509104.

[12] Supriyo Datta. *Quantum Transport: Atom to Transistor*. Cambridge University Press, Cambridge, 2005. ISBN 978-1-139-16431-3. doi: 10.1017/CBO9781139164313.

[13] Robert H Dennard, Fritz H Gaensslen, Hwa-Nien Yu, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of solid-state circuits*, 9(5):256–268, 1974. ISSN 0018-9200.

[14] Florin-Silviu Dumitru, Nicoleta Cucu-Laurenciu, Alexandru Matei, and Marius Enachescu. Graphene Nanoribbons Based 5-Bit Digital-to-Analog Converter. *IEEE Transactions on Nanotechnology*, 20:248–254, 2021. ISSN 1941-0085. doi: 10.1109/TNANO.2021.3063602.

[15] Elizabeth J. Duplock, Matthias Scheffler, and Philip J. D. Lindan. Hallmark of Perfect Graphene. *Physical Review Letters*, 92(22):225502, June 2004. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.92.225502.

[16] A. El Gamal. Trends in CMOS image sensor technology and design. In *Digest. International Electron Devices Meeting,*, pages 805–808, December 2002. doi: 10.1109/IEDM.2002.1175960.

[17] Lawrence J. Fogel and Alvin J. Owens. Artificial intelligence through simulated evolution, 1966.

[18] Yohan Frans, Jaewook Shin, Lei Zhou, Parag Upadhyaya, Jay Im, Vassili Kireev, Mohamed Elzeftawi, Hiva Hedayati, Toan Pham, Santiago Asuncion, Chris Borrelli, Geoff Zhang, Hongtao Zhang, and Ken Chang. A 56-Gb/s PAM4 Wireline Transceiver Using a 32-Way Time-Interleaved SAR ADC in 16-nm FinFET. *IEEE Journal of Solid-State Circuits*, 52(4):1101–1110, April 2017. ISSN 1558-173X. doi: 10.1109/JSSC.2016.2632300.

[19] A. E. Galashev and O. R. Rakhmanova. Mechanical and thermal stability of graphene and graphene-based materials. *Physics-Uspekhi*, 57(10):970, October 2014. ISSN 1063-7869. doi: 10.3367/UFNe.0184.201410c.1045.

[20] A. K. Geim and K. S. Novoselov. The rise of graphene. *Nature Materials*, 6(3):183–191, March 2007. ISSN 1476-4660. doi: 10.1038/nmat1849.

[21] Frank Gray. Pulse code communication. *United States Patent Number 2632058*, 1953.

[22] Jon Guerber. Flash ADC. https://commons.wikimedia.org/wiki/File:Flash_ADC.png, December 2009.

[23] Wilfried Haensch, Tayfun Gokmen, and Ruchir Puri. The Next Generation of Deep Learning Hardware: Analog Computing. *Proceedings of the IEEE*, 107(1):108–122, January 2019. ISSN 1558-2256. doi: 10.1109/JPROC.2018.2871057.

[24] John Hennessy and David Patterson. A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 27–29, June 2018. doi: 10.1109/ISCA.2018.00011.

[25] D. Hisamoto, T. Kaga, Y. Kawamoto, and E. Takeda. A fully depleted lean-channel transistor (DELTA)-a novel vertical ultra thin SOI MOSFET. In *International Technical Digest on Electron Devices Meeting*, pages 833–836, December 1989. doi: 10.1109/IEDM.1989.74182.

[26] Miao Hu, Catherine E. Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, Noraica Davila, Hao Jiang, R. Stanley Williams, J. Joshua Yang, Qiangfei Xia, and John Paul Strachan. Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine. *Advanced Materials*, 30(9):1705914, 2018. ISSN 1521-4095. doi: 10.1002/adma.201705914.

[27] Tyler W. Hughes, Ian A. D. Williamson, Momchil Minkov, and Shanhui Fan. Wave physics as an analog recurrent neural network. *Science Advances*, 5(12):eaay6946, December 2019. doi: 10.1126/sciadv.aay6946.

[28] Y. Jiang, N. Cucu Laurenciu, and S. D. Cotofana. On Carving Basic Boolean Functions on Graphene Nanoribbons Conduction Maps. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, Florence, May 2018. IEEE. ISBN 978-1-5386-4881-0. doi: 10.1109/ISCAS.2018.8351421.

[29] Y. Jiang, N. Cucu Laurenciu, and S.D. Cotofana. Non-Equilibrium Green Function-based Verilog-A Graphene Nanoribbon Model. In *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*, pages 1–4, July 2018. doi: 10.1109/NANO.2018.8626396.

[30] Yande Jiang. *Graphene-Based Computing: Nanoribbon Logic Gates & Circuits*. PhD thesis, Delft University of Technology, Delft, December 2020.

[31] Yande Jiang, Nicoleta Cucu Laurenciu, and Sorin Cotofana. Complementary Arranged Graphene Nanoribbon-based Boolean Gates. In *2018 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–7, July 2018.

[32] Yande Jiang, Nicoleta Cucu Laurenciu, and Sorin Dan Cotofana. On Basic Boolean Function Graphene Nanoribbon Conductance Mapping. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(5):1948–1959, May 2019. ISSN 1549-8328, 1558-0806. doi: 10.1109/TCSI.2018.2882310.

[33] Theodore Kaczynski. Industrial Society and its Future. *The Washington Post*, page 38, September 1995.

[34] Sung-Jin Kim, Zachary Myers, Steven Herbst, Byongchan Lim, and Mark Horowitz. 20-GS/s 8-bit Analog-to-Digital Converter and 5-GHz Phase Interpolator for Open-Source Synthesizable High-Speed Link Applications. *IEEE Solid-State Circuits Letters*, 3:518–521, 2020. ISSN 2573-9603. doi: 10.1109/LSSC.2020.3037823.

[35] T. Kuroda. CMOS design challenges to power wall. In *Digest of Papers. Microprocesses and Nanotechnology 2001. 2001 International Microprocesses and Nanotechnology Conference (IEEE Cat. No.01EX468)*, pages 6–7, October 2001. doi: 10.1109/IMNC.2001.984030.

[36] Changgu Lee, Xiaoding Wei, Jeffrey W. Kysar, and James Hone. Measurement of the Elastic Properties and Intrinsic Strength of Monolayer Graphene. *Science*, 321(5887):385–388, July 2008. doi: 10.1126/science.1157996.

[37] H. Lee, L.-E. Yu, S.-W. Ryu, J.-W. Han, K. Jeon, D.-Y. Jang, K.-H. Kim, J. Lee, J.-H. Kim, S. Jeon, G. Lee, J. Oh, Y. Park, W. Bae, H. Lee, J. Yang, J. Yoo, S. Kim, and Y.-K. Choi. Sub-5nm All-Around Gate FinFET for Ultimate Scaling. In *2006 Symposium on VLSI Technology, 2006. Digest of Technical Papers.*, pages 58–59, June 2006. doi: 10.1109/VLSIT.2006.1705215.

[38] Max C. Lemme, Tim J. Echtermeyer, Matthias Baus, and Heinrich Kurz. A Graphene Field-Effect Device. *IEEE Electron Device Letters*, 28(4):282–284, April 2007. ISSN 0741-3106. doi: 10.1109/LED.2007.891668.

[39] Ming Li, Yanhui Liu, and Y. Jay Guo. Shaped Power Pattern Synthesis of a Linear Dipole Array by Element Rotation and Phase Optimization Using Dynamic Differential Evolution. *IEEE Antennas and Wireless Propagation Letters*, 17(4):697–701, April 2018. ISSN 1548-5757. doi: 10.1109/LAWP.2018.2812816.

[40] Mingyun Lv, Jun Li, Huafei Du, Weiyu Zhu, and Junhui Meng. Solar array layout optimization for stratospheric airships using numerical method. *Energy Conversion and Management*, 135: 160–169, March 2017. ISSN 0196-8904. doi: 10.1016/j.enconman.2016.12.080.

[41] Jiin-Jang Maa and Ching-Yuan Wu. A new constant-field scaling theory for MOSFET's. *IEEE Transactions on Electron Devices*, 42(7):1262–1268, July 1995. ISSN 1557-9646. doi: 10.1109/16.391208.

[42] Sally A. McKee. Reflections on the memory wall. In *Proceedings of the 1st Conference on Computing Frontiers*, CF '04, page 162, New York, NY, USA, April 2004. Association for Computing Machinery. ISBN 978-1-58113-741-5. doi: 10.1145/977091.977115.

[43] Jianhua Mo, Ahmed Alkhateeb, Shadi Abu-Surra, and Robert W. Heath. Hybrid Architectures With Few-Bit ADC Receivers: Achievable Rates and Energy-Rate Tradeoffs. *IEEE Transactions on Wireless Communications*, 16(4):2274–2287, April 2017. ISSN 1558-2248. doi: 10.1109/TWC.2017.2661749.

[44] Gordon E Moore. Cramming more components onto integrated circuits. 38(8):6, 1965.

[45] Dong-Ryeol Oh. A 6-Bit 20 GS/s Time-Interleaved Two-Step Flash ADC in 40 nm CMOS. *Electronics*, 11(19):3052, January 2022. ISSN 2079-9292. doi: 10.3390/electronics11193052.

[46] Yusuke Oike and Abbas El Gamal. CMOS Image Sensor With Per-Column ΣΔ ADC and Programmable Compressed Sensing. *IEEE Journal of Solid-State Circuits*, 48(1):318–328, January 2013. ISSN 1558-173X. doi: 10.1109/JSSC.2012.2214851.

[47] B. Partoens and F. M. Peeters. From graphene to graphite: Electronic structure around the K point. *Physical Review B*, 74(7):075404, August 2006. doi: 10.1103/PhysRevB.74.075404.

[48] Alain Petrowski and Sana Ben-Hamida. *Evolutionary Algorithms*. John Wiley & Sons, April 2017. ISBN 978-1-119-13638-5.

[49] Dirk Pfaff, Xin-Jie Wang, Chai Palusa, Robert Abbott, Shahaboddin Moazzeni, Leisheng Gao, Mei-Chen Chuang, Rolando Ramirez, Maher Amer, and Tae Young Goh. A 56-Gb/s Long-Reach Fully Adaptive Wireline PAM-4 Transceiver in 7-nm FinFET. *IEEE Solid-State Circuits Letters*, 2 (12):285–288, December 2019. ISSN 2573-9603. doi: 10.1109/LSSC.2019.2953840.

[50] Henry H. Radamson, Huilong Zhu, Zhenhua Wu, Xiaobin He, Hongxiao Lin, Jinbiao Liu, Jinjuan Xiang, Zhenzhen Kong, Wenjuan Xiong, Junjie Li, Hushan Cui, Jianfeng Gao, Hong Yang, Yong Du, Buqing Xu, Ben Li, Xuewei Zhao, Jiahan Yu, Yan Dong, and Guilei Wang. State of the Art and Future Perspectives in Advanced CMOS Technology. *Nanomaterials*, 10(8):1555, August 2020. ISSN 2079-4991. doi: 10.3390/nano10081555.

[51] Ingo Rechenberg. Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment Library Translation 1122*, 1965.

[52] Kilian Roth, Hessam Pirzadeh, A. Lee Swindlehurst, and Josef A. Nossek. A Comparison of Hybrid Beamforming and Digital Beamforming With Low-Resolution ADCs for Multiple Users and Imperfect CSI. *IEEE Journal of Selected Topics in Signal Processing*, 12(3):484–498, June 2018. ISSN 1941-0484. doi: 10.1109/JSTSP.2018.2813973.

[53] Majid Sepahvand, Fardin Abdali-Mohammadi, and Farhad Mardukhi. Evolutionary Metric-Learning-Based Recognition Algorithm for Online Isolated Persian/Arabic Characters, Reconstructed Using Inertial Pen Signals. *IEEE Transactions on Cybernetics*, 47(9):2872–2884, September 2017. ISSN 2168-2275. doi: 10.1109/TCYB.2016.2633318.

[54] Adam Slowik and Halina Kwasnicka. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 32(16):12363–12379, August 2020. ISSN 1433-3058. doi: 10.1007/s00521-020-04832-8.

[55] Warren D Smith. Fundamental physical limits on computation. *available from ResearchIndex< http://citeseer. nj. nec. com/smith95fundamental. html*, 29, 1995.

[56] Young-Woo Son. Energy Gaps in Graphene Nanoribbons. *Physical Review Letters*, 97(21), 2006. doi: 10.1103/PhysRevLett.97.216803.

[57] R.H. Walden. Analog-to-digital converter survey and analysis. *IEEE Journal on Selected Areas in Communications*, 17(4):539–550, April 1999. ISSN 1558-0008. doi: 10.1109/49.761034.

[58] H. Wang, N. Cucu Laurenciu, Y. Jiang, and S. D. Cotofana. Atomistic-Level Hysteresis-Aware Graphene Structures Electron Transport Model. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2019. doi: 10.1109/ISCAS.2019.8702106.

[59] H. Wang, N. Cucu Laurenciu, Y. Jiang, and S. D. Cotofana. Ultra-Compact, Entirely Graphene-Based Nonlinear Leaky Integrate-and-Fire Spiking Neuron. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, October 2020. doi: 10.1109/ISCAS45731.2020.9181092.

[60] He Wang, Nicoleta Cucu Laurenciu, Yande Jiang, and Sorin Cotofana. Graphene-Based Artificial Synapses with Tunable Plasticity. *ACM Journal on Emerging Technologies in Computing Systems*, 17(4):50:1–50:21, June 2021. ISSN 1550-4832. doi: 10.1145/3447778.

[61] Darrell Whitley. An overview of evolutionary algorithms: Practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831, December 2001. ISSN 0950-5849. doi: 10.1016/S0950-5849(01)00188-4.

[62] John M Wincn. CMOS-transistor-based digital-to-analog converter. January 1987.

[63] Xinyu Wu, Vishal Saxena, Kehan Zhu, and Sakkarapani Balagopal. A CMOS spiking neuron for brain-inspired neural networks with resistive synapses and in situ learning. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(11):1088–1092, 2015. ISSN 1549-7747.

[64] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, March 1995. ISSN 0163-5964. doi: 10.1145/216585.216588.

[65] Benwei Xu, Yuan Zhou, and Yun Chiu. A 23-mW 24-GS/s 6-bit Voltage-Time Hybrid Time-Interleaved ADC in 28-nm CMOS. *IEEE Journal of Solid-State Circuits*, 52(4):1091–1100, April 2017. ISSN 1558-173X. doi: 10.1109/JSSC.2016.2642204.

[66] Xuejun Yang, Zhiyuan Wang, Jingling Xue, and Yun Zhou. The Reliability Wall for Exascale Supercomputing. *IEEE Transactions on Computers*, 61(6):767–779, June 2012. ISSN 1557-9956. doi: 10.1109/TC.2011.106.

[67] Xinjie Yu and Mitsuo Gen. *Introduction to Evolutionary Algorithms*. Springer Science & Business Media, June 2010. ISBN 978-1-84996-129-5.

[68] Minglei Zhang, Yan Zhu, Chi-Hang Chan, and Rui P. Martins. A 20GS/s 8b Time-Interleaved Time-Domain ADC with Input-Independent Background Timing Skew Calibration. In *2021 Symposium on VLSI Circuits*, pages 1–2, June 2021. doi: 10.23919/VLSICircuits52068.2021.9492436.