Informatics

Waveform Prediction of Digital Circuits by Sigmoidal Approximation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Josef Salzmann
Matrikelnummer 11777724

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Ulrich Schmid
Mitwirkung: Univ.Ass. Dipl.-Ing. Dipl.-Ing. Dr. Jürgen Maier
Dipl.-Ing. Arman Ferdowsi

Wien, 29. Februar 2024

Josef Salzmann	Ulrich Schmid

Informatics

Waveform Prediction of Digital Circuits by Sigmoidal Approximation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Josef Salzmann
Registration Number 11777724

to the Faculty of Informatics
at the TU Wien

Advisor: Univ.Prof. Dr. Ulrich Schmid
Assistance: Univ.Ass. Dipl.-Ing. Dipl.-Ing. Dr. Jürgen Maier
Dipl.-Ing. Arman Ferdowsi

Vienna, 29th February, 2024

Josef Salzmann   Ulrich Schmid

Erklärung zur Verfassung der Arbeit

Josef Salzmann

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29. Februar 2024

Josef Salzmann

Danksagung

Ich möchte mich bei meinem Betreuer Ulrich Schmid für seine Beratung und die ergebnisreichen Diskussionen bedanken, die bei der Durchführung dieser Diplomarbeit essentiell waren. Danke an Jürgen Maier und Arman Ferdowsi, deren konstruktiver Kritik zur Arbeit beigetragen hat. Ich danke meiner Familie, ganz besonders meinen Eltern, auf deren bedingungslose Unterstützung ich meine ganze Studienzeit bauen konnte.

Diese Arbeit wurde im Kontext des FWF Projektes DMAC (P32431) durchgeführt.

Acknowledgements

I want to thank my advisor Ulrich Schmid for his guidance and the fruitful discussions, which were essential in the creation of this thesis. Thanks to my advisors Jürgen Maier and Arman Ferdowsi, who provided me with constructive criticism throughout the course of this thesis. I would like to thank my family, especially my parents, on whose unconditional support I could rely on throughout my studies.

This work was conducted in the context of the FWF project DMAC (P32431).

Kurzfassung

In der Entwicklung digitaler Systeme ist die Analyse des zeitlichen Verhaltens unabdingbar. Herkömmlicherweise geschieht diese durch Simulation auf analoger oder digitaler Ebene. Analoge Simulatoren wie SPICE lösen die Gleichungen, die die Komponenten einer Schaltung beschreiben, iterativ. Obzwar in der erreichbaren Genauigkeit unübertroffen, skaliert diese Methode äußerst schlecht, und kann nur für Systeme überschaubarer Größe eingesetzt werden. Demgegenüber stehen digitale Simulatoren, welche auf Kosten der Genauigkeit die analogen Vorgänge weitestgehend vernachlässigen. Herkömmliche digitale Simulatoren sind in der Lage, Systeme praktisch unbegrenzter Größe zu simulieren. Diese Diplomarbeit präsentiert einen Simulationsansatz, der in die Lücke zwischen beiden Extremen fällt: Die analogen Signale digitaler Schaltungen werden durch parametrierte Sigmoide angenähert. Da Sigmoide sowohl Zeitpunkte der Schwellwertüberschreitungen als auch Flankensteilheiten abbilden können, kann das Ausgangsverhalten von Invertern und NOR Gattern durch Übertragungsfunktionen für diese Parameter vorhergesagt werden. Durch die Nutzung künstlicher neuronaler Netze(ANN) arbeitet dieser Ansatz wesentlich schneller als ein analoger Simulator und bietet gleichzeitig mehr Genauigkeit als ein digitaler Simulator. Unsere Prototyp-Implementierung erzielt vielversprechende Ergebnisse bezüglich der Simulationsgeschwindigkeit und Genauigkeit.

Abstract

Investigating the temporal behavior of digital circuits is a crucial step in digital system design, usually done via either analog or digital simulation. Analog simulators like SPICE iteratively solve the differential equations characterizing the circuits' components. Although unrivalled in its accuracy, this method is only feasible for small designs, as its drawback is the high computational effort. Digital timing simulators are based on a digital abstraction of the analog behavior of a circuit. In state of the art tools, propagation of signals along cells is predicted using pre-computed delay values. This method can handle designs of virtually any size, but has the drawback of sacrificing accuracy. This thesis presents another approach, which aims to fill the gap in-between the two existing ones: We use parameterized sigmoids for approximating the analog waveforms appearing in digital circuits. Since sigmoids allow to conveniently encode both threshold crossing times and steepness, we can predict output waveforms of inverters and `NOR` gates by determining how these parameters are transferred. Harnessing the power of artificial neural networks (ANN), this novel approach operates substantially faster than an analog simulator, while offering better accuracy than a digital simulator. Promising results regarding accuracy and simulation speed obtained by our implemented prototype demonstrate the potential of our approach.

Contents

CHAPTER 1

Introduction

Modern digital systems usually consist of several millions if not billions of transistors. Viewed at the lowest level of implementation, the elementary components of digital integrated circuits like inverters, NOR gates and other gates are analog electronic devices. The digital abstraction is created by digitizing the waveforms via some threshold voltage.

During the development of such systems, behavioral and timing simulations have to be performed to ensure proper functioning. Consequently, simulating digital circuits to estimate the propagation of signals throughout the circuit is a crucial task in the development process of digital systems. State-of-the-art tools performing these simulations mainly use two approaches, namely, analog and digital simulations.

The former approach simulates digital circuits using analog simulators like SPICE [1], which use highly sophisticated physical models like BSIM [2, 3]. Current and voltage trajectories are simulated by iteratively solving the differential equations describing the circuit components. Although these simulations yield highly accurate results, their high consumption of time and computational resources renders them impractical for digital circuits consisting of a large number of components.

The latter approach discretizes the analog signals of the circuit to their respective digital equivalent of true (1) and false (0). Consequently, digital components are characterized by the input-to-output delay values of rising/falling transitions under various conditions. Instead of solving differential equations at simulation time, the simulator uses tables of pre-computed delay values [4, 5] to parameterize pure or inertial delay channels [6]. Prominent examples of this approach are the current source models (CSM) Effective CSM (ECSM)[4] and Composite CSM (CCSM)[5]. Both models conduct analog simulations to generate tables of voltage (ECSM) or current (CCSM) traces for different input slopes and output capacitance values. To simulate a circuit with these approaches, the surrounding of each cell has to be analyzed beforehand to retrieve the fitting table entry.

Static timing analysis allows the accurate corner case analysis of circuits consisting of a large number of components, but does not allow to determine the delay of individual transitions. Digital dynamic timing analysis approaches have been invented to enable this: Generalizing the concept of the inertial delay channel, the so-called Delay Degradation Model (DDM) has been developed in [7, 8]. This model acknowledges the fact that pulses propagating a gate will shorten, and eventually stop propagating, if the pulse length is below a certain threshold. This is an improvement over inertial delay channels, where pulses either propagate fully or not at all. The DDM is an instance of a single history model, where the delay for a given input transition depends on the time difference of the input to the previous output. However, in [9] it has been shown that the delay functions used in the DDM, and all other existing models, do not match reality (are unfaithful). This problem is avoided by the Involution Delay Model (IDM) proposed in [10], which is based on delay formulas that are self-inverse, i.e., involutions. Based on the IDM, several models were developed, with some of them also incorporating multi-input switching (MIS) effects, also known as the Charlie effect [11] [10, 12, 13].

In this thesis, we will advocate an approach that is somewhat in-between analog and digital simulation. As detailed in Section 1.2, we will use parametrized sigmoids for representing the analog waveforms appearing in digital circuits.

1.1 Background

The fundamental operations of digital systems are realized by the interplay of combinatorial blocks, implementing Boolean formulas, and storage elements. Although the combinatorial parts of digital systems can be described by Boolean formulas, e.g., $Z = \neg I$ or $Z = A \vee B$, Boolean algebra is not intended to be used to analyze the concrete implementation of digital systems. Since physical implementations of Boolean functions will most likely never operate infinitely fast [14] (a maximum frequency of 1PHz is suggested to be the upper bound for optoelectronics), the temporal behaviour of these implementations has to be investigated to ensure correctness.

Note that every Boolean formula can be rewritten using only the `NOR` operator, hence it is called functionally complete. This justifies why we can solely focus on the `NOR` gate in our proof of concept. Of course, real world applications will employ a wide range of different gates apart from `NOR` gates. These gates can be modeled in a similar way as the `NOR` gate, however.

There are various ways to construct digital systems, possibilities range from mechanical [15], biological [16] to the more prominent electronic systems. Since the majority of digital systems are electronic circuits, mainly CMOS [17] nowadays, this thesis will solely focus on those. Figure 1.1 shows how an inverter and a `NOR` gate are implemented using transistors in CMOS technology.

A CMOS inverter is built using one pMOS and one nMOS transistor and works as follows: If I is connected to V_{DD} (digital 1), the nMOS conducts current and hence connects the output Z to GND (digital 0), while the pMOS is non-conducting and does not affect the output. If I is connected to GND the roles of pMOS and nMOS reverse and Z is connected to V_{DD}.

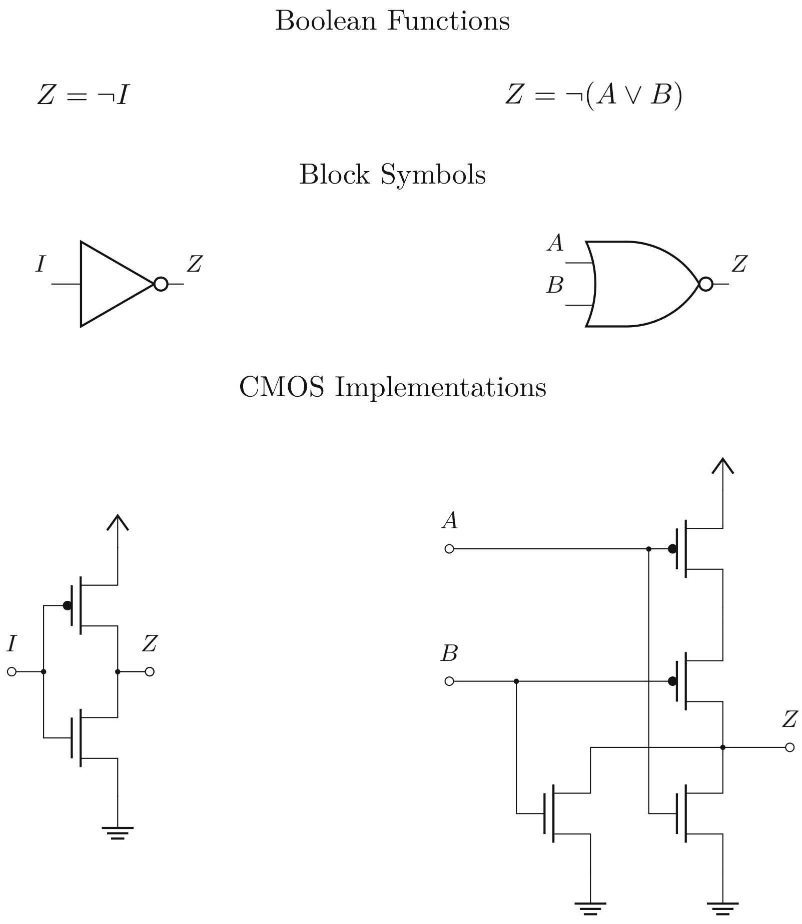


Figure 1.1: Inverter and `NOR` gate.

More complex gates can be built through the use of several nMOS and pMOS transistors, like in the CMOS implementation of a `NOR` gate also depicted in Figure 1.1.

1.2 Main Contribution

The main contribution of this thesis is to develop a model that simulates digital circuits by sigmoidal approximations of analog waveforms. The goal is to predict output transitions by using the information of the current input transition and the preceding output transition. In that sense, it can be thought of as generalization of the IDM presented in [9]. While the IDM has its main focus on faithfulness, the presented model is more concerned about the average prediction performance and neglects faithfulness.

Building on the idea of single history channels, one can increase the information supplied to the model by not only supplying the time difference between the current input and the most recent output transition, but also information about the slope (steepness). Sigmoid waveforms are the most natural candidates to achieve this. We will present the so called third order model (TOM), which uses the time difference between the last output and current input transition and the slope information of these two transitions to predict the delay and slope information of the resulting output transition. The cornerstones of this thesis are:

Sigmoidal approximation of digital waveforms: We present a method of approximating digital waveforms by sums of sigmoids. This approximation allows us to encode the information of a waveform as a list of reals, which can be interpreted as lossy compression.

Output waveform prediction through ANNs: Based on the representation of digital waveforms as lists of reals, we investigate how an inverter and a `NOR` gate transform their input lists into output lists. Using ANNs as universal function approximators, we can establish gate specific transfer functions to calculate output parameter sequences based on input parameter sequences.

Implementation of a prototype: As a proof of concept and to investigate the capabilities of our model, we implemented a prototype simulator, which is publicly available on github [18].

Note that ANNs are not the only means for output prediction. Naturally, polynomials, splines and lookup tables are also candidates. While polynomials are computationally easy to handle, they suffer from low accuracy. On the other hand, lookup tables predict with reasonable accuracy but are too slow due to their size. When lookup tables are used, virtually every lookup will have to reach to main memory, since the lookup table is too large to be cached; this is also the reason why using splines is not feasible. ANNs, on the other hand, are compact and, interestingly, offer even better accuracy than lookup tables. Therefore, we found that the best choice was to use ANNs.

1.3 Related Work

In [19], circuit cells are modeled as resistor-resistor-capacitor cells. Transitions are modeled as segments of exponential charge/discharge curves, having their start/endpoints in between GND and V_{DD}.

A somewhat similar approach to this thesis can be found in [20]. Using a data set consisting of SPICE simulations, an ANN is trained to predict the output delay and transition time, based on the input rise time. It should be noted, though, that the authors focus more on the case of multi-input switching (MIS). This means that the ANN takes the transition times of two inputs, rather than one, and the time difference between them, as input parameters for its prediction. Additionally, this model also uses the output capacitance of the respective gate as prediction parameter.

Although many different approaches for digital logic simulation exist [21, 22], we are not aware of any other existing work that directly relates to the approach proposed in this thesis.

1.4 Thesis Structure

The main ideas of our presented approach will be presented in Chapter 2 and Chapter 3. Chapter 4 outlines some insights into the implementation of the presented approach. The achieved results are discussed in Chapter 5. Some shortcomings and remarks are collected in Chapter 6. Finally, Chapter 7 concludes the thesis.

CHAPTER 2

Third Order Model

This chapter will introduce the core ideas and assumptions the presented approach relies on. Its focus will be the way of approximating digital waveforms by parametrized sigmoids and the use of these approximations for waveform prediction. As in the case of other timing prediction models [6, 7, 20, 23], our third order model (TOM) also operates on the basis of transitions. But instead of characterizing transitions by their rise/fall time or their steepness at $\frac{V_{DD}}{2}$, we will use the gathered fitting parameters of our fitting function.

2.1 Sigmoidal Approximation of Waveforms

Let us state the definition of a sigmoid according to [24]: A sigmoid function is a bounded differentiable real function that is defined for all inputs values and that has a positive derivative everywhere.

Building on the fact that any arbitrary function can be approximated arbitrarily well by adding sigmoids [25], we will use sigmoids to approximate the voltage waveforms observed in digital circuits. Obviously, the choice of the particular sigmoid will directly affect the fitting quality we will be able to achieve. In Figure 2.1, several different sigmoids are depicted. Although all of them approach the same limits and their derivative at $x = 0$ is 1, they differ in shape. Note that the list of sigmoids depicted in Figure 2.1 is by no means exhaustive: One can easily build new sigmoids by, e.g., linear combinations of two existing ones or other modifications. The authors' bachelor thesis [26] presents one way of adjusting the curvature of a given sigmoid using Taylor series.

In this thesis, we will focus solely on the logistics function as a means of approximating the waveforms of digital circuits. Our TOM thus approximates real waveforms by a sum of appropriately parameterized sigmoids according to Equation (2.1). Two parameters determine the shape of the logistics function: The parameter a tells us the *steepness* of the transition, while the parameter b tells us *when* the transition happens.

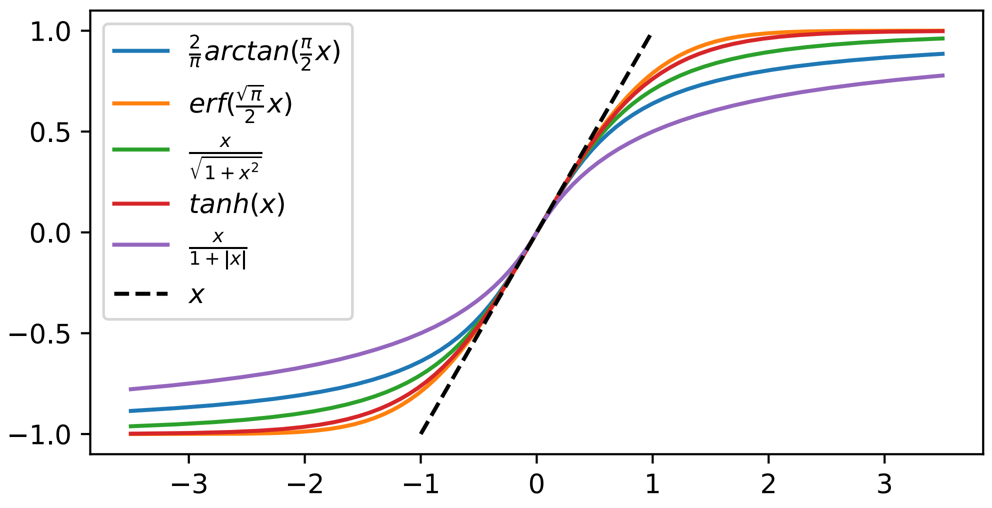


Figure 2.1: Different Sigmoids.

$$F(t, a, b) = \frac{1}{1 + e^{-a(t-b)}} \tag{2.1}$$

In Section 3.2, we will provide the detailed procedure for determining the list of parameters representing a waveform. Even though sigmoids of different shapes could be constructed to better match real waveforms, the focus of this thesis is to present a proof of concept of the general approach.

2.2 Gate Output Signal Prediction

As illustrated in Figure 2.2, every transition is characterized by a tuple (s, t), where s represents the slope parameter eq. (2.1) and t the shift parameter b. In Figure 2.2 $f(s_{in}, t_{in})$ is used as an abbreviation of $F(t, s_{in}, t_{in})$ to stress that only s_{in} and t_{in} are of interest. The basic idea for modeling signal propagation in the TOM is to use the tuples characterizing the current input transition (s_{in}, t_{in}) and the previous output transition $(s_{o_{n-1}}, t_{o_{n-1}})$ to predict the tuple characterizing the generated output transition. This is expressed in the following equation:

$$(s_{o_n}, t_{o_n} - t_{in}) = F_G(t_{in} - t_{o_{n-1}}, s_{in}, s_{o_{n-1}}) \tag{2.2}$$

where F_G is a gate dependent transfer function. For each input transition, we are now able to compute the corresponding output transition. Given F_G and a list of input transition parameters, Algorithm 2.1 gives an overview of how we can implement this for a single gate. Note that for the first transition at the input we introduced a sentinel previous transition with parameters $(s, -\infty)$, where the sign of s depends on the sign of

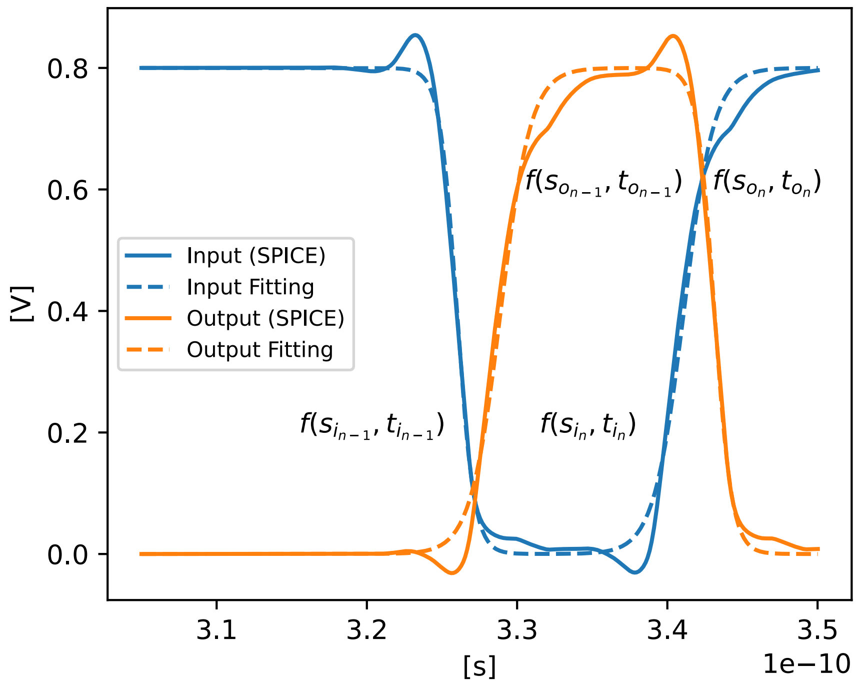


Figure 2.2: Fitting a waveform using sigmoids in the TOM.

the first input transition. The absolute value of s is not of much interest and can be set to any reasonable value. Since this sentinel transition happens at time $-\infty$, it will not have any influence on the prediction of the first output transition anyway. It is just there to ensure that we can treat the first input transition like any other transition.

2.3 Cancellation

One crucial detail missing in Algorithm 2.1 is the possibility of a transition not being propagated due to cancellation. Figure 2.3 depicts an example, where two input transitions are too close to each other and hence do not propagate to the output. Although the input waveform crosses the threshold $\frac{V_{DD}}{2}$ twice, the corresponding output waveform does not cross this threshold anywhere. This means that the two tuples describing these transitions, (s_{i_n}, t_{i_n}) and $(s_{i_{n+1}}, t_{i_{n+1}})$, do not result in corresponding output tuples (s_{o_n}, t_{o_n}) and $(s_{o_{n+1}}, t_{o_{n+1}})$, i.e., should not be propagate at all. Note however, that the output waveform of Figure 2.3 could be fitted to yield such output parameters. We do not consider these parameters to be worth the effort, however, since sub-threshold spikes do not propagate through a digital gate in general anyway.

Algorithm 2.1: Base Pseudo-Code for Output parameter prediction for gate G.

```
    Input: List of input parameter tuples $((s_{i_n}, t_{i_n}))_{n \geq 1}$, sorted by ascending $t_{i_n}$
    Output: List of output parameter tuples
 1  add $(-\infty, s)$ to Output;
 2  Prev $\leftarrow (-\infty, s)$;
 3  while $(s_{i_n}, t_{i_n}) \in$ Input ascending in time do
 4      $(s_{o_{n-1}}, t_{o_{n-1}}) \leftarrow$ Prev;
 5      T $\leftarrow t_{i_n} - t_{o_{n-1}}$;
 6      if $s_{i_n} > 0$ then
 7          $(s_{o_n}, t'_{o_n}) \leftarrow F_G^{\uparrow}(\mathrm{T}, s_{o_{n-1}}, s_{i_n})$;
 8      else
 9          $(s_{o_n}, t'_{o_n}) \leftarrow F_G^{\downarrow}(\mathrm{T}, s_{o_{n-1}}, s_{i_n})$;
10      end
11      $t_{o_n} \leftarrow t'_{o_n} + t_{i_n}$;
12      Prev $\leftarrow (s_{o_n}, t_{o_n})$;
13      add $(s_{o_n}, t_{o_n})$ to Output;
14  end
```

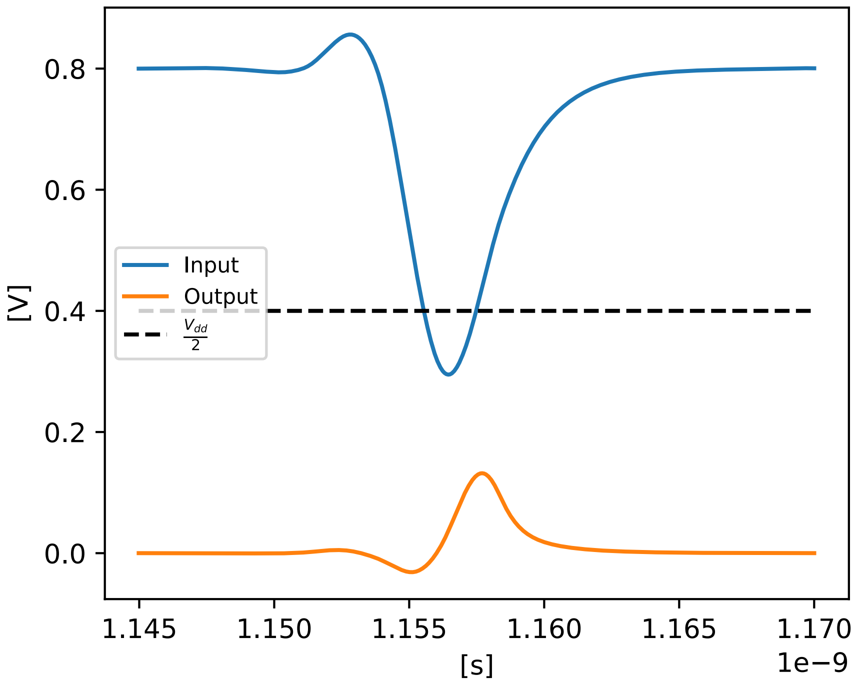


Figure 2.3: Example of pulse cancellation.

Algorithm 2.2: Pseudo-Code of TOM with Cancellation.

```
Input: List of input parameter tuples $((s_{i_n}, t_{i_n}))_{n \geq 1}$, sorted by ascending $t_{i_n}$
Output: List of output parameter tuples
1  add $(-\infty, s)$ to Output;
2  Prev ← $(-\infty, s)$;
3  while $(s_{i_n}, t_{i_n}) \in$ Input ascending in time do
4      $(s_{o_{n-1}}, t_{o_{n-1}}) \leftarrow$ Prev;
5      T ← $t_{i_n} - t_{o_{n-1}}$;
6      if $s_{i_n} > 0$ then
7          $(s_{o_n}, t'_{o_n}) \leftarrow F_G^{\uparrow}(\mathrm{T}, s_{o_{n-1}}, s_{i_n})$;
8      else
9          $(s_{o_n}, t'_{o_n}) \leftarrow F_G^{\downarrow}(\mathrm{T}, s_{o_{n-1}}, s_{i_n})$;
10     end
11     $t_{o_n} \leftarrow t'_{o_n} + t_{i_n}$;
12     if $Cancellation(s_{o_{n-1}}, t_{o_{n-1}}, s_{o_n}, t_{o_n})$ then
13         remove $(s_{o_{n-1}}, t_{o_{n-1}})$ from Output;
14         $(s_{o_{n-2}}, t_{o_{n-2}}) \leftarrow$ last element of Output;
15         Prev ← $(s_{o_{n-2}}, t_{o_{n-2}})$;
16     else
17         Prev ← $(s_{o_n}, t_{o_n})$;
18         add $(s_{o_n}, t_{o_n})$ to Output;
19     end
20 end
```

We add a mechanism to Algorithm 2.1, which takes care of such cancellations and works as follows: Since a cancellation can only happen between two adjacent transitions, it is enough check the current and the previous output transition with parameters (s_{o_n}, t_{o_n}) and $(s_{o_{n-1}}, t_{o_{n-1}})$. We verify whether the waveform built out of $f(s_{o_n}, t_{o_n}) + f(s_{o_{n-1}}, t_{o_{n-1}})$ crosses $\frac{V_{DD}}{2}$ at any point. If this is not the case we treat these two transitions as canceled, otherwise nothing changes.

Although the additional case distinction added in Algorithm 2.2 may look innocent, it has subtle implications for the resulting implementation. More specifically, it can happen that the latest transition that has been added to “Output” will be canceled by the next incoming transition. Since the “Input” and “Output” queues of two connected gates are basically the same, it could be the case that the successor gate has already processed a transition that gets canceled. For such cases, the implementation has to supply appropriate data structures that handle the deletion of canceled transitions along the propagation path.

CHAPTER 3

Transfer Relation

This chapter gives an overview of how we establish a relationship between the TOM input and output parameters of logic gates, and its implementation in an ANN that can be used for prediction later on. We also present our approaches for generating the training data, the fitting process and the architecture of the ANN, used in our implemented tool. Restricting our attention to logic gates with fan-out of exactly one, we instantiate our approach for both an inverter and a `NOR` gate.

3.1 Data Generation via SPICE Simulations

The following setup was chosen to obtain the data for building our model. An inverter chain made up of an arbitrary number n of inverters $G_1, ..., G_n$, preceded by four pulse-shaping stages also consisting of inverters, see Figure 3.1, is stimulated by a sequence of zero-time (Heaviside) transitions. An example of zero-time transitions used as stimulus and the input of the inverter G_1 after the pulse-shaping section is shown in Figure 3.3. Pulse-shaping is necessary, since we only want to add data to our data set that results from physically possible waveforms. The end of the inverter-chain is terminated by two inverters in series, where the final one drives a capacitor. This ensures that G_n also has its output connected to an inverter. For our simulations, we used the 15 nm FinFet models of the Nangate Open Cell Library [27].

In this circuit, the inputs of G_1 to G_n and the output of G_n are recorded. The length of the chain n can be chosen according to the input stimulus. Indeed, to save time and memory, pulse trains that vanish after a few gates can be recorded using circuits with a small n. At the same time, n can be chosen bigger if the pulses propagate through many gates. In our experiments, n was usually around 20.

Although not drawn in Figure 3.1, the input and output of two adjacent stages are connected by a parasitic network. For ease of data generation, we modified the generated parasitic network between two inverters by the place and route tool, and plugged the

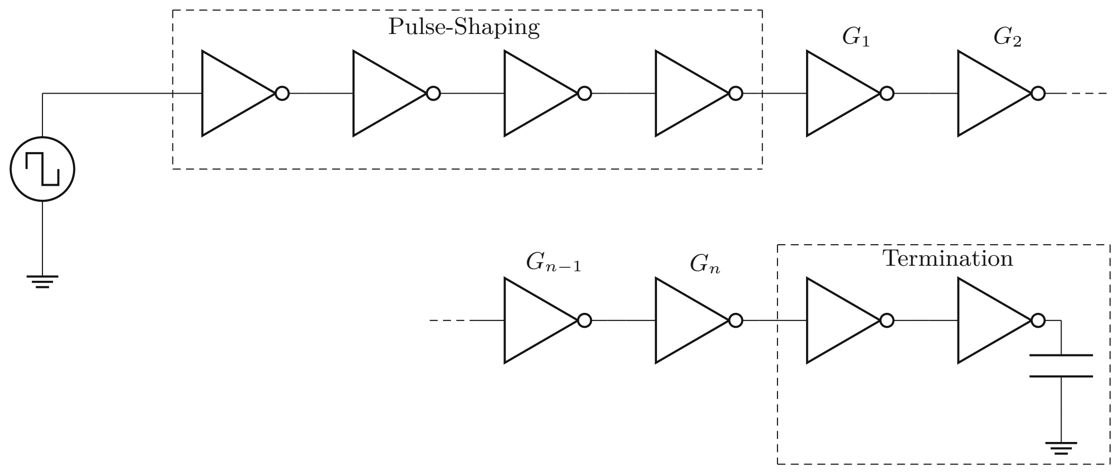
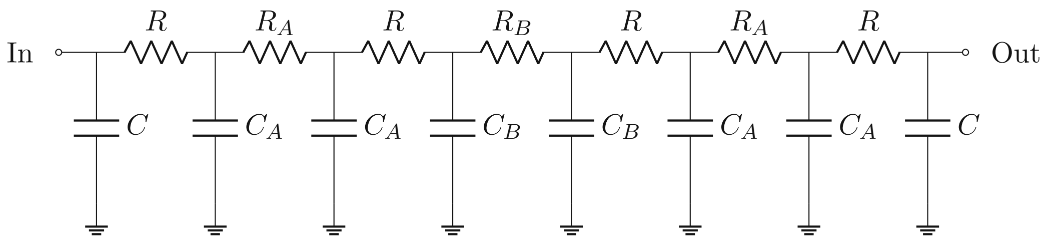


Figure 3.1: Inverter-chain used for Data Generation.



$R = 4\,\Omega$	$C = 0.033\,35\,\text{fF}$
$R_A = 0.173\,714\,\Omega$	$C_A = 0.026\,16\,\text{fF}$
$R_B = 2.258\,29\,\Omega$	$C_B = 0.091\,13\,\text{fF}$

Figure 3.2: Parasitic Network connecting two adjacent Stages.

network shown in Figure 3.2 between every two stages of the inverter chain. Note that using the same network everywhere allows us to treat all stages of the inverter chain as having the *same* input output relation with respect to our model.

To cover all the possible waveforms a circuit can encounter at its input, the zero-time transitions of the stimulus are generated in a systematic way. The three time intervals T_A, T_B, and T_C, shown in Figure 3.3, are varied in such a way that the coverage is maximized. We used the following procedure here: As a first step, all time intervals are set to large values, such that, at the output of the pulse-shaping section, the voltage in between two transitions definitely stays at V_{DD} resp. GND for some time. This yields the upper bounds $T_{AU} = T_{BU} = T_{CU} = 25\text{ps}$ for the respective time intervals.

The lower bounds are obtained by lowering one of the three values until only two transitions are generated at the output of the pulse-shaping section. This yields the lower bound $T_{AL} = T_{BL} = T_{CL} = 3\text{ps}$ for the respective time intervals.

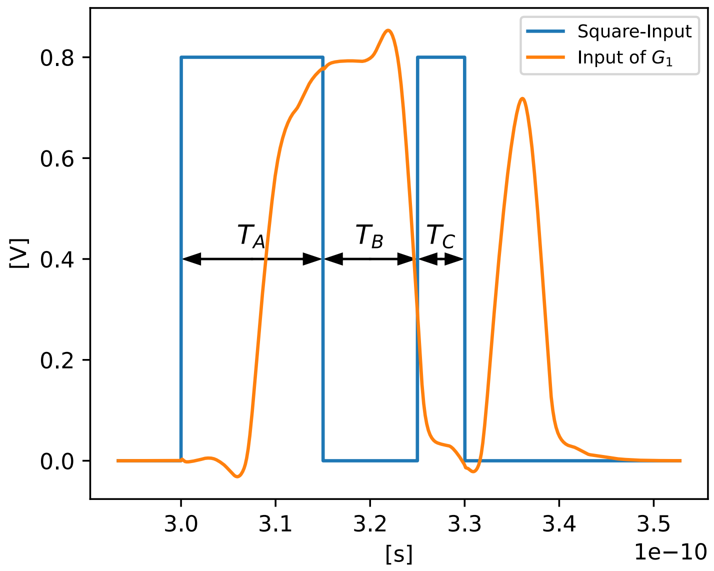


Figure 3.3: Pulse-shaping input and output example.

Having gathered the upper and lower bounds for the respective time intervals, we then constructed the set of stimulus parameters. We subdivided each time interval into reasonable sub-intervals and defined a set containing the boundaries of these sub-intervals. For instance, the set for T_A is given by:

$$S_{TA} = \{3\text{ps}, 3.5\text{ps}, 4\text{ps}, ..., 24.5\text{ps}, 25\text{ps}\}, \tag{3.1}$$

which was constructed by dividing $[T_{AL}, T_{AU}]$ into approximately 50 sub-intervals. The sets S_{TB} and S_{TC} were constructed analogously. Choosing the granularity of these sets is a trade-off between available simulation time and desired data set size. A fine grained set implies that the resulting training data set is densely populated, but also means that a considerable number of analog simulations has to be conducted. Note that in our case all three sets S_{TA}, S_{TB}, and S_{TC} are the same, since they consist of the same elements, but in general this need not be the case. With the three sets S_{TA}, S_{TB}, and S_{TC}, the complete set of stimulus vectors S was constructed:

$$S = S_{TA} \times S_{TB} \times S_{TC} \tag{3.2}$$

S contains approximately $50^3 = 125000$ interval combinations. For each of those, we ran analog simulations using Spectre and recorded the simulation files. Each such waveform was finally fitted to our TOM, according to the procedure explained in the next subsection.

3.2 Waveform Fitting Process

We use the well-known least squares fitting method [28] to match a model function to a given waveform in our data sets. Formally speaking, the least squares fitting method aims to minimize the metric

$$\sum_{i=1}^{m} \left[\frac{y(t_i) - F(t_i, \boldsymbol{x})}{\sigma_i} \right]^2 \tag{3.3}$$

where m is the cardinality of the data set consisting of tuples (t_i, y_i) representing $y_i(t_i)$, $F(t_i, \boldsymbol{x})$ is the model function, $\boldsymbol{x}$ the vector of parameters, and σ_i provides a means of assigning a specific weight to the ith data point. For our fittings we use the `curve_fit` implementation of scipy[1].

3.2.1 Single Transition Model Function

The choice of an adequate model function is crucial to ensure a proper fitting of the obtained data sets. As already mentioned in Chapter 2, we use the logistic function [24, 29] given in Equation (3.4) for this purpose:

$$F(t) = \frac{1}{1 + e^{-t}} \tag{3.4}$$

Although any other sigmoid shown in Figure 2.1 would probably also meet our requirements, previous experience in [26] has shown that the logistic function is very well suited for our approach. As Equation (3.4) does not supply any parameters for the fitting algorithm to work on, we enhanced it as follows:

$$F(t, a, b) = \frac{1}{1 + e^{-a(t-b)}} \tag{3.5}$$

With the two parameters a and b, we can vary the shape of the logistic function according to our needs. With $a > 0$, a rising transition can be modeled, while a falling one can be represented with $a < 0$. The absolute value of a will determine how fast a transition happens. The parameter b allows us to place the transition at any point of time we desire.

Although Equation (3.5) already supplies us with the necessary parameters for fitting, we actually use a slightly modified version, given by:

$$F_s(t, a, b) = \frac{1}{1 + e^{-a(t \cdot 10^{10} - b)}} \tag{3.6}$$

The multiplication factor 10^{10} added to t ensures that both parameters a and b are in the same range: Since the time periods we work on are in the range of picoseconds, omitting this multiplicative factor would result in a being in the range of 10^{12}, while b would be in the range of 10^{-12}. This would be rather inconvenient for visualization and processing, however.

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html`

3.2.2 General Model Function

In general, a waveform consists of several transitions. In that case, it is not possible to isolate and fit each transition on its own. All transitions have to be fitted at the same time to capture the effects two neighboring transitions cause to each other. For a waveform consisting of m transitions, we consider the following joint model function:

$$F_g(t, x_0, ..., x_{2\cdot m-1}) = V_{DD}\left(\sum_{i=0}^{m-1} F_s(t, x_{2\cdot i}, x_{2\cdot i+1}) \; - F_{Comp}(x_0, m)\right), \tag{3.7}$$

where $(x_{2\cdot i}, x_{2\cdot i+1})$ are the parameters of the ith transition. The function F_{comp}, given in Equation (3.8), guarantees that F_g will result in a function that stays between GND and V_{DD}. Subtracting F_{comp} is needed, since adding an arbitrary number of single transition model functions would generally result in a function value between $k \cdot V_{DD}$ and $(k+1)\cdot V_{DD}$.

$$F_{Comp}(x_0, m) = \left\lfloor \frac{m}{2} \right\rfloor - C(x_0, m) \tag{3.8}$$

$$C(x_0, m) = \begin{cases} 1 & \text{if } \; x_0 < 0 \text{ and } m\%2 = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

3.2.3 Initial Guesses and Bounds

For good fitting quality, the fitting algorithm must be supplied with reasonable bounds and initial guesses for the parameters we wish to obtain from fitting. In order to generate these guesses and bounds, we will resort to heuristics. The main idea is to use the second derivative $\frac{\partial^2 V(t)}{(\partial t)^2}$ of the waveform $V(t)$ to identify the inflection points. These inflection points will serve as initial guesses for the shift parameters, while the initial guesses for the steepness are set to a reasonable constant value for each transition. Overall, the bounds will be set generously yet firmly enough such that the fitting algorithm does not change the transition order determined by the heuristics.

In the following, we will describe the heuristics we chose for generating the initial guesses and bounds in more detail. While the guesses and bounds for the steepness parameters are not too difficult to generate, determining the shift parameters is a bit subtle. More precisely, for the steepness parameter we only need to know whether we identified a rising or a falling transition and choose a typical steepness value as an initial guess, while setting the lower/upper bound about one magnitude smaller/larger. For example, in our case the steepness parameters for a falling transition were usually around -180. Consequently, the lower bound was to -2000 and the upper bound to -20. These bounds are supplied to the fitting algorithm.

For the shift parameter we use the zero-crossings $\frac{\partial^2 V(t)}{(\partial t)^2} = 0$ of the second derivative to identify the inflection points. From the value of the first derivative at the inflection points, we can infer whether it is a rising or falling transition: if $\frac{\partial V(t)}{\partial t} > 0$ it is a rising transition, if $\frac{\partial V(t)}{\partial t} < 0$, it is a falling transition. However, since $V(t)$ also contain over-

and undershoots, as can be seen in Figure 3.3, we have to distinguish between *valid* and non-valid inflection points. To achieve this, we introduce a lower/upper bound $\chi = 0.05\,\mathrm{V}$. The value $V(t)$ at valid inflection points must lie between χ and $V_{DD} - \chi$. Having identified all inflection points that meet these criteria, we can use them as initial guesses for the shift parameters. The respective bounds are set to lie in-between two adjacent inflection points. For example, if the nth inflection point is located at t_n, the initial value for the nth shift parameter will be t_n, while the lower bound will be $\frac{t_{n-1}+t_n}{2}$ and the upper bound $\frac{t_n+t_{n+1}}{2}$.

3.2.4 Usage of σ

During our attempts to fit the voltage waveforms in our simulation data, we observed that some transitions that only differ by a marginal numerical noise resulted in parameter values that differed by a considerable amount. This means that the fitting algorithm was caught in different local minima in its search space, which might severely impose the prediction performance of our method. To reduce the number of close-by local minima, i.e., guide the fitting algorithm towards our desired fitting, we chose to use the weight σ in way that the region around an inflection point adds more to the error term. The region of interest around an inflection point is defined in the following way: Starting from an inflection point we move along the time axis in both directions until either $V(t)$ leaves the range between $V_{DD} - \chi = 0.8\,\mathrm{V} - 0.05\,\mathrm{V} = 0.75\,\mathrm{V}$ and $\chi = 0.05\,\mathrm{V}$ or $\frac{\partial V(t)}{\partial t}$ changes sign.

To guide the fitting algorithm, the region around inflection points will have a σ value lower than 1, meaning that these regions contribute more to the error term. The choice of how much bigger this contribution should be is delicate. If σ is too small the fitting quality outside of these regions will be poor, while σ too big means that maybe the number of local minima did not decrease. In our case we chose σ to be around $\frac{1}{25}$ inside of these regions. By making σ depend on the first derivative $\frac{\partial V(t)}{\partial t}$ inside the inflection point region, see Figure 3.4, we express that the fitting should focus on the inflection point and gradually return to $\sigma = 1$ as we move to the border of the inflection point region.

3.2.5 Waveform Clipping

As typical waveforms contain over- and undershoots, which cause $V(t)$ to rise above V_{DD} or fall below *GND*, the approach to fit these waveforms with sigmoids faces some difficulties: Since a single sigmoid cannot model such over- and undershoots the fitting algorithm will calculate relatively big error terms, which may lead to a degradation of the fitting process. In order to mitigate this problem, we intentionally clip away over- and undershoots, meaning that $V(t)$ is always between V_{DD} and *GND*.

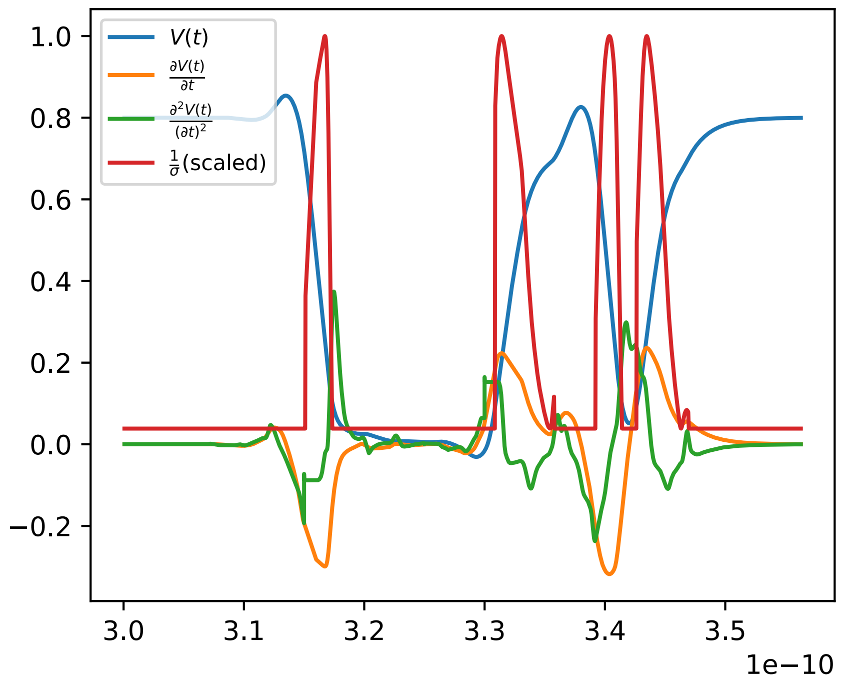


Figure 3.4: σ Visualization. Inside the inflection point region $\frac{1}{\sigma}$ is scaled according to $\frac{\partial V(t)}{\partial t}$.

3.3 Transition Relation Generation

The previous sections explained our approach of fitting a waveform and representing it as a sequence of parameters $((s_n, t_n))$. In Algorithm 2.2, we presented our algorithm for predicting the output parameter sequence of a gate, given an input parameter sequence. Algorithm 2.2 uses the two transition functions $F_G^{\uparrow}(\mathrm{T}, s_{o_{n-1}}, s_{i_n})$ and $F_G^{\downarrow}(\mathrm{T}, s_{o_{n-1}}, s_{i_n})$ for prediction. We now present how we generated the ANNs $F_G^{\uparrow}$ and $F_G^{\downarrow}$ from simulation data.

3.3.1 Simulation Data to Training Data

The data gathered from the simulation of the circuit shown in Figure 3.1 consists of several thousand waveforms. All these waveforms are fitted using Equation (3.7) and their fitting parameters $x_0, ..., x_{2 \cdot m-1}$ are stored. Based on these parameters we trained our ANNs $F_G^{\uparrow}$ and $F_G^{\downarrow}$. The whole process is explained by the example of Figure 3.5, which depicts the output of three gates of the inverter chain and the respective fittings. The fitting parameters of transitions $t_1, ..., t_4$ are listed in Table 3.1.

The basic idea is that the output of gate G_n is the result of its input, which is the output of G_{n-1}. For instance, looking at Table 3.2, transition t_3 of G_1 will result in transition t_3 at G_2, hence $F_G^{\uparrow}(3.417 - 3.319, 119.350, 99.378)$ should yield (-184.887, $3.438 - 3.417$). Two more equivalences are listed in Table 3.2, which represent the training set for $F_G^{\uparrow}$. The training set for $F_G^{\downarrow}$, which takes the parameter of falling input transitions as input, is constructed analogous. Of course, the full tables containing all the training data generated from the simulations are huge, they consist of several hundred thousand entries.

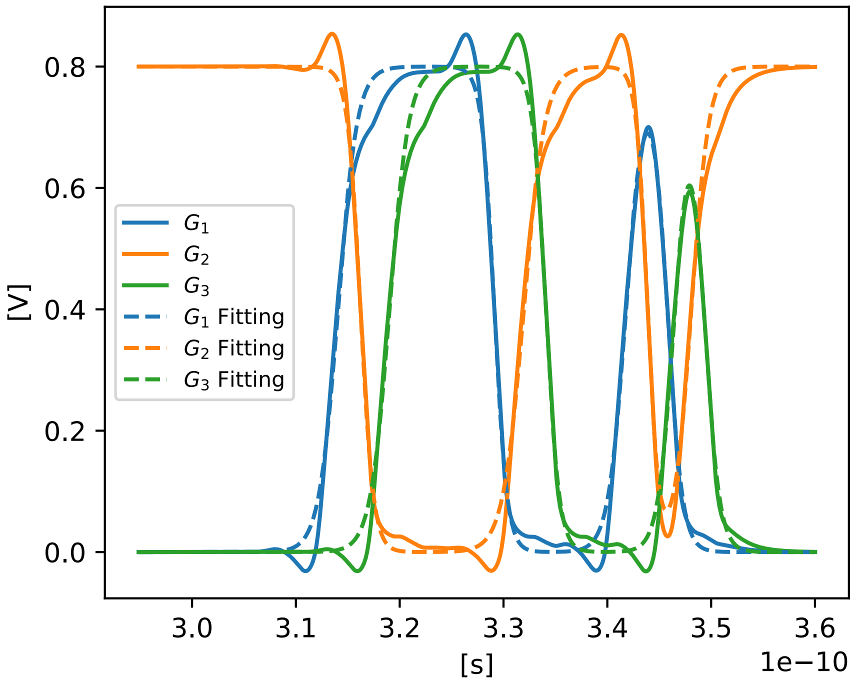


Figure 3.5: Example waveforms with their fittings.

Output Node	t_1	t_2	t_3	t_4
G_1	(109.213, 3.140)	(-164.174, 3.291)	(119.350, 3.417)	(-141.482, 3.458)
G_2	(-161.041, 3.162)	(99.378, 3.319)	(-184.887, 3.438)	(120.154, 3.480)
G_3	(109.033, 3.190)	(-156.254, 3.340)	(131.668, 3.464)	(-139.814, 3.493)

Table 3.1: Fitting values for Figure 3.5.

3.3.2 ANN Topology

We used TensorFlow [30] for training our ANNs. For an inverter, we need to generate 4 ANNs: Both $F_G^{\uparrow}$ and $F_G^{\downarrow}$ are implemented by two ANNs, one that predicts the output transition steepness s_{o_n} and one for the output shift (delay) d, where $d = t_{o_n} - t_{i_n}$.

T	s_{i_n}	$s_{o_{n-1}}$	s_{o_n}	d
0.0982	119.350	99.378	-184.887	0.0208
0.1293	99.378	109.033	-156.254	0.0212
0.0165	120.154	131.668	-139.814	0.0124

Table 3.2: Example Parameter Table generated from Table 3.1. Herein, $T = t_{i_n} - t_{o_{n-1}}$ and $d = t_{o_n} - t_{i_n}$.

For the topology of our ANNs we chose a network with 3 internal layers, where the first two layers have 10 neurons and the last one has 5, all layers use relu activation functions. Since there are three input neurons and one output neuron the topology is given by [3,10,10,5,1]. The prediction accuracy of this architecture is around 1%. We experimented with several other ANN architectures but did not conduct a systematic search. Future work may investigate more advantageous architectures.

3.4 Data Generation for **NOR** Gates

So far, this chapter was only concerned with inverters. Since we also want to cover gates with more than one input, we also characterized a NOR gate. The procedure for a gate with more than one input stays the same, only the SPICE simulations, of course, have to be conducted with the desired gate. In Figure 3.6, the circuit for SPICE simulations for a NOR gate is depicted. As in the case of inverters, the parasitics in between two adjacent gates were set to those in Figure 3.2. Note that this procedure has to be done for every input. With respect to the NOR gate, this means that we also need to run simulations with the other input. In the case of NAND gates, for example, one would have to tie the other input to V_{DD} as opposed to GND in Figure 3.6 to construct an inverter chain to record the desired data.

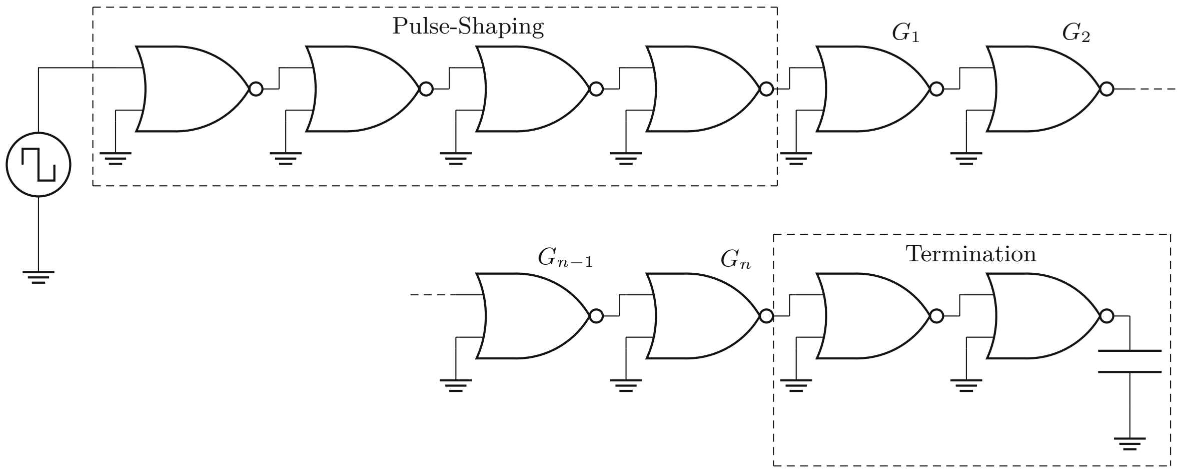


Figure 3.6: Inverter-chain of NOR gates used for Data Generation.

3.5 Valid Region Containment

No matter how good the prediction of our ANNs (or any other means of prediction) is, we always have to expect some error. If we look at a single gate, this error is to be expected and lies in the nature of an approximation. But if we look at a chain of gates, where one gate is the input of the next gate, we have to anticipate that the erroneous prediction of a gate will lead to an even more erroneous prediction of the successor gate. In other words, the predicted parameters after a few gates may have diverged so far that no reasonable prediction can be expected anymore.

Especially ANNs can show arbitrary behavior if they are presented with input values that are far outside of the training set. Suppose, across a chain of gates, only one prediction causes the parameters to fall outside this training set. In that case, the predictions based on these faulty parameters will become arbitrary and, therefore, meaningless. Unless countermeasures are taken, this flaw will render our approach useless.

Our solution to this problem works as follows: Since the problem arises from the fact that an ANN may be called with input values for which it was never trained, we have to ensure that this can never happen. This means that we have to identify the region for which our ANNs were trained, in the following we call this the "valid region".

3.5.1 Computing a Valid Region

Since we already generated the data to train our ANNs, we use this data to compute a hull separating an inner valid region and an outer invalid region. Of course, this entails that the training data covers all possible parameter combinations that appear in a physical waveform. Since we have three inputs, T, s_{i_n} and $s_{o_{n-1}}$ as depicted in Table 3.2, this region will have three dimensions. Populating the three dimensional Euclidean space with our data points (T, s_{i_n}, $s_{o_{n-1}}$), we can compute the concave hull of this point cloud, which will yield such a valid region. Care has to be taken with this operation, since the concave hull, unlike the convex hull, is not uniquely defined in general [31]. The algorithm we use to compute the concave hull, supplied by pcl [32], offers a parameter α that essentially controls the granularity of the resulting concave hull. Setting α to a small value yields a fine grained hull, which will lead to good separation of the valid and invalid regions but comes at the cost of high computational effort during run time. Setting α to a large value has the opposite effects. We chose $\alpha = 20$ which, looking at Figure 3.7, seems a reasonable compromise.

3.5.2 Usage of Valid Region

The computed valid region from above allows us to ensure that the ANNs are always invoked with parameters that lie inside the training data. Since an input to an ANN (T, s_{i_n}, $s_{o_{n-1}}$) can again be interpreted as a point in three dimensional Euclidean space we can easily test whether or not it is inside our computed valid region. This now allows the following case distinction:

(i) The input values are inside the valid region: In this case, nothing has to be done since the problem we stated does not arise.

(ii) The input values are outside of the valid region: In this case, we have to replace the invalid input values (T, s_{i_n}, $s_{o_{n-1}}$) with the closest point (T^v, $s^v_{i_n}$, $s^v_{o_{n-1}}$) that lies on the hull defining the valid region.

It should be noted that care has to be taken regarding the scale of the single parameters T, s_{i_n} and $s_{o_{n-1}}$. Since the library [33] we used to implement our operations uses the Euclidean distance, i.e., treats all three dimensions equal when computing the distance, the correction will be flawed if not all three parameters are approximately scaled the same. In our case, the steepness parameters s_{i_n} and $s_{o_{n-1}}$ are in the range of ~100, while T can be in the range of ~0.1. If this would not be changed, the Euclidean distance will be dominated by the steepness parameters, which would result in the closest point (T^v, $s^v_{i_n}$, $s^v_{o_{n-1}}$) having an arbitrary value for T^v. To ensure that the Euclidean distance takes all three dimensions into account, we can stretch the dimension of T such that it will have the same scale as the other parameters. For this purpose we multiply T with 10^4 for computing (T^v, $s^v_{i_n}$, $s^v_{o_{n-1}}$) and divide the T^v that resulted from the correction by 10^4 to return back to normal parameter range. Figure 3.7 depicts an example of how the input parameters (T, s_{i_n}, $s_{o_{n-1}}$) populate three dimensional space, in this picture the parameter T is multiplied by a factor of 500 for ease of visualization.

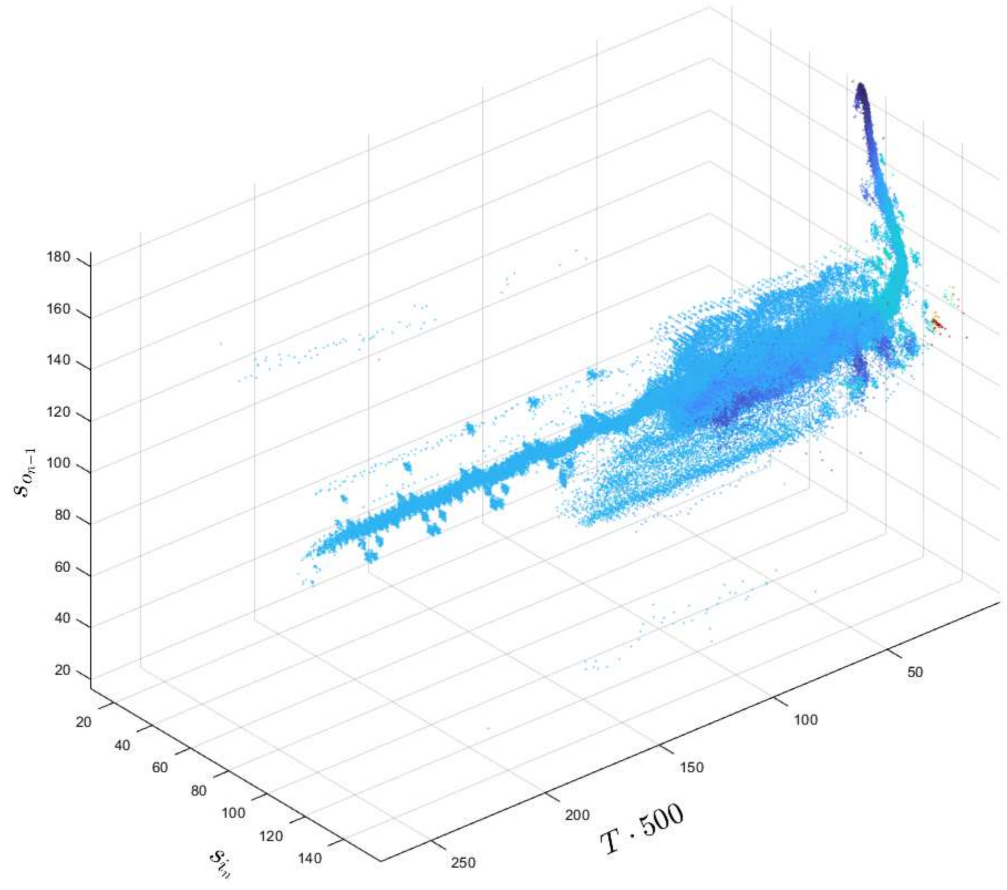


Figure 3.7: Example of how (T, s_{i_n}, $s_{o_{n-1}}$) populate three the dimensional space.

CHAPTER 4

Implementation

This chapter gives an overview of the prototype implementation of a simulation tool for digital circuits, which utilizes the transfer function-based approach described in the previous chapters. Our framework has been implemented in C++ and is available on github [18]. Since this is only a prototype implementation, for ease of usage, we assumed that every `NOR` gate and every inverter have the same transfer function F_G. This way, only two transfer functions have to be supplied to the tool, see Section 4.2.5. Of course, under realistic circumstances, virtually every gate has a different transfer function F_G, caused by different fan-out and parasitics.

4.1 Basic Tool Operation

In order to conduct a simulation, the tool has to be invoked with a file containing the description of the circuit, links to stimulus waveform files, and the ANN files. A description of theses files can be found in Section 4.2.

As a first action, the program loads the specified ANNs and builds its internal representation of the given circuit specification. After this, the initial values of all gate outputs are computed, which is non-trivial if the circuit contains feedback loops. Then, based on the stimulus waveforms, the transition schedule is computed and executed.

4.1.1 Initial Values

In order to properly simulate a circuit, the initial values of each gate output have to be determined at the start of the simulation. Since not all circuits posses a well-defined initial state (e.g., an odd-numbered inverter ring), the tool must check if it exists. In order to answer this question and to determine appropriate initial values, we use a SAT-solver. Of course, to simplify the implementation, we could have relied on the user to supply correct initial states for each gate to tool. But this is a tedious and error prone process, which also becomes non trivial when circuits with feedback are at hand. Our tool hence provides a way to automatically compute feasible initial states. Note that (the existence of) a well-defined initial state not only depends on the circuit, but also on externally supplied initial values. For example, Figure 4.1 depicts a ring of three `NOR` gates with one external input. If the input is set to GND, the ring oscillates, whereas with the input set to V_{DD}, it has a well-defined state.

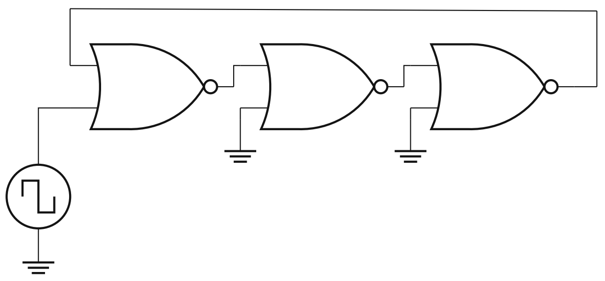


Figure 4.1: Three `NOR` gates in a ring.

Therefore, the circuit in Figure 4.1 can only be simulated if the first transition of the external input is falling. Furthermore, as mentioned when presenting Algorithm 2.1, in order to propagate the first input transition of a gate, we added a transition $(-\infty, s)$ to the output list. To determine the sign of s, we have to compute the initial state of the gate. If the initial output of a gate is GND, s will be negative since we pretend that a falling transition took place at $t = -\infty$; if the initial output is V_{DD}, we pretend that a rising transition took place at $t = -\infty$. The absolute value of s is simply set to a fixed value, since it does not affect the calculation of the first output transitions anyway.

In the following, we will describe our approach for constructing a formula representing a circuit and the initial values of each input. For ease of understanding, Figure 4.2 depicts the labeling we use for our formulas. For each `NOR` gate and inverter, a single variable is

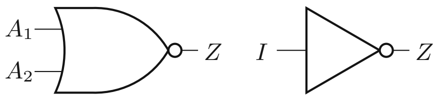


Figure 4.2: Gate Labeling.

used to describe the respective output; $\top$ represents true, $\perp$ represents false. All that remains is to state the input/output relation of each gate. For a `NOR` gate, this relation is given by:

$$Z \iff \neg(A_1 \lor A_2) \tag{4.1}$$

which can be converted to a conjunctive normal form (CNF) as follows, see e.g., [34] for details:

$$(A_1 \lor A_2 \lor Z) \land (\neg A_1 \lor \neg Z) \land (\neg A_2 \lor \neg Z) \equiv \top \tag{4.2}$$

The relation for an inverter, on the other hand, is given by:

$$Z \iff \neg I \tag{4.3}$$

which converted to CNF yields:

$$(I \lor Z) \land (\neg I \lor \neg Z) \equiv \top \tag{4.4}$$

If an input A_1, A_2 or I is directly connected to GND/V_{DD} or to an external input, it is set to the respective Boolean value. If an input is connected to the output of another gate G_n, the formula describing this input will be the respective variable V_n that describes the output of G_n.

The clause set supplied to the SAT-solver will consist of an instance of Equation (4.2) for every `NOR` gate and an instance of Equation (4.4) for every inverter. If the SAT-solver determines that the clause set is unsatisfiable, the circuit does not have a well-defined initial state and cannot be simulated. If the clause is satisfiable, the SAT-solver will return the corresponding variable assignment. From this assignment, we can derive the initial state of each gates' output[1].

4.1.2 Transition Schedule

After the initial values have been determined, the actual simulation can be executed. Right after initialization, the transitions of all inputs are read in and stored in a global transition schedule, where the transitions are ordered according to their shift parameter in ascending order. Each transition does contain not only its fitting parameters, but also metadata like its source and its sink(s) or the parent and child transitions, which is important in the case of cancellation.

As long as there are transitions in the schedule, the transition with the smallest shift parameter will be removed, and Algorithm 2.2 will be performed on it. Since a transition at the input of a gate may result in a transition at the output of a gate, the generated output transitions are added to the transition schedule.

4.1.3 Simulation Output

In the tool input file, the user can specify the gate outputs that should be recorded; of course, the output of every gate in the circuit can be recorded. After all transitions are processed and the transition schedule is empty, the output transition of the gates specified in the input file are written to an output file.

4.2 Tool Input Files

The program is called with a main file that specifies the input waveforms, the circuit to be simulated, the circuit nodes to be recorded, and the references to the ANNs and valid regions to be used. This main references other files like input stimulus and files containing the transfer relations. Lines that start with $\star$ are comments and will be skipped.

[1]It should be noted that, in general, care has to be taken with the initial values of stateful elements, e.g., latches. There might be constellations where both GND and V_{DD} are valid initial values for these elements.

4.2.1 Stimulus File Format

Since our tool operates on the basis of transitions, and every waveform can be represented as a sequence of transitions, the stimulus is represented as a sequence of transition parameters. For example, the file `InA.csv` with the content:

```
-197.5899086, 10.4053535
 201.5360181, 10.6480647
-192.6446782, 10.6819359
```

contains three transitions. Although the transitions do not have to be listed in temporal order, it is strongly recommended to do so.

4.2.2 Inputs

Starting with the keyword `INPUTS`, this section lists the files containing the stimulus waveforms and the external input identifiers that can be used in the circuit specification. An example of this section will look like:

```
INPUTS
INA InA.csv
```

Each input has to be separated by a new line.

4.2.3 Circuit Specification

After the keyword `GATES`, the circuit to be simulated is specified. So far, two elements can be used to construct circuits, namely 2-input `NOR` gates, and inverters. A `NOR` gate can be specified by the following line:

```
NOR GATE_NAME NOR_OUPUT_NAME INPUTA_NAME INPUTB_NAME
```

where every keyword ending on `_NAME` is an input resp. output identifier that can be chosen freely. Likewise, an inverter can be specified by the following line:

```
INV INV_NAME INV_OUTPUT_NAME INPUT_NAME
```

In addition to external input identifiers, the keywords `GND` and `VDD` can be used to serve as inputs.

4.2.4 Outputs

After the keyword `OUTPUTS`, a list of gate output identifiers that should be recorded can be specified.

4.2.5 Transfer Relations

At last, the paths to the ANNs and the valid region files have to be provided. Recall that an output tuple (s_{o_n}, t_{o_n}) is calculated by $F_G(t_{i_n} - t_{o_{n-1}}, s_{i_n}, s_{o_{n-1}})$, see Equation (2.2). Since F_G is divided in two parts, namely $F_G^{\uparrow}$ and $F_G^{\downarrow}$, both have to be specified. $F_G^{\uparrow}$ and $F_G^{\downarrow}$ are provided by two different ANNs each, one that calculates t_{o_n} (shift) and one that calculates s_{o_n} (steepness).

This section begins with the keyword `TRANSFERFUNCTIONS` and is followed by the list of the respective ANNs, valid region files and their paths. For example:

```
SIS_A_F ANN tfs/SIS_A_falling_model
SIS_A_F OFF tfs/SIS_A_falling_input.off
```

specifies that the ANNs that predict s_{o_n} and t_{o_n} are located in `./tfs/SIS_A_falling_model_shift` and `./tfs/SIS_A_falling_model_steepness`, while the valid region file is `./tfs/SIS_A_falling_input.off`. The complete list of all ANNs and region files that have to specified is found in the examples in the public github repository [18].

4.3 Usage

To invoke the simulator, only the above specified input file has to be provided through the option `-c`. Example invocations and input files can be found in the public github repository [18].

4.4 Dependencies

Table 4.1 lists the direct dependencies of the implementation.

CppFlow	`https://github.com/serizba/cppflow`
Cryptominisat 5.6.8	`https://github.com/msoos/cryptominisat`
plog	`https://github.com/SergiusTheBest/plog`
pcl 1.12.1	`https://pointclouds.org/`
cgal 5.4	`https://www.cgal.org/`

Table 4.1: Table of direct dependencies.

CppFlow is able to load pre-trained Tensorflow models from C++ code, enabling us to easily train our ANNs in Python and execute them in C++. Cryptominisat provides a generic SAT-solver used for determining the initial conditions, as explained in Section 4.1.1. plog serves as generic logging library. The Point Cloud Library (pcl) provides a function for computing the concave hull of a three dimensional point cloud.

The Computational Geometry Algorithms Library (cgal) is used to determine whether or not a three dimensional point lies inside a concave hull, as depicted in Section 3.5.2.

CHAPTER 5

Evaluation

In this chapter, we will present some quantitative results we achieved using the approach presented in this thesis. By comparing the predictions of our model to the predictions of SPICE, we were able to shed some light on the achievable accuracy. Since we ran the SPICE simulations on a dedicated server equipped with the required licenses, while our tool ran on a local computer, a fair running time comparison is impossible. The vastly different computing platforms cause a severe bias in the time measurements, which is also dependent on the simulated circuit. Additionally, the SPICE settings allow to reduce simulation accuracy, which also impacts SPICE simulation time. We therefore avoid stating speed-up numbers for our approach.

5.1 Error Metric

Accuracy comparison is a delicate task in our setting, since the output of our tool is neither an analog waveform nor a series of binary signal transitions. In order to compare the prediction of our tool with SPICE, we compare the root-mean-square error (RMSE) of the area under the predicted trace and the SPICE waveform. There are two variations that are of interest in our case: We will compare our prediction both to the plain SPICE waveform, and to the TOM fitting of the SPICE waveform. In Figure 5.1, an example waveform with its optimal fitting and the prediction is depicted. Comparing the prediction to the optimal fitting allows us to inspect how far off the prediction is from the best possible results, since we cannot achieve a better prediction than the optimal fitting. By comparing the SPICE waveform to our prediction, we can evaluate the overall prediction and fitting error, which gives an insight on how well our approach performs.

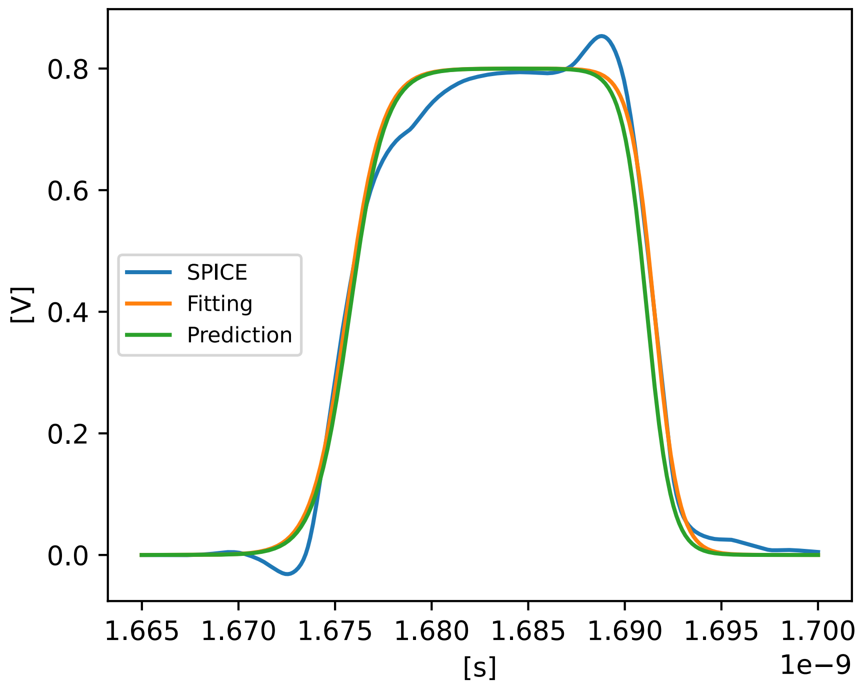


Figure 5.1: Example SPICE waveform with its optimal fitting and prediction.

5.2 Inverter Chain Example

The simplest circuit to test our approach on is the one that was used to generate the training data, i.e., an inverter chain made of NOR gates, see Figure 3.1. We expect that the prediction of our approach will have the least error possible on this circuit and therefore gives a lower bound on the achievable accuracy. The inverter chain we used has hundred stages. After each stage, we compare the prediction of our approach with both the SPICE waveform and the fitting of the SPICE waveform. The input of the inverter chain is generated by a pulse-shaping section consisting of 10 inverters, the pulse-shaping input is stimulated by a random transition sequence of length 50, with Gaussian distribution of the inter-transition times with mean $\mu = 10\,\text{ps}$ and variance $\sigma = 5\,\text{ps}$. We considered the average of 200 such sequences.

Table 5.1 lists the first few elements of such a trace, which were calculated by the fitting algorithm from the respective SPICE waveform. Table 5.2 lists the prediction for the input trace given in Table 5.1. The Input row of both tables is the same, since this is the stimulus. Figure 5.2 depicts an example waveform taken from a stage of the inverter chain. The predicted waveform is compared to the fitting of the SPICE waveform and the SPICE waveform itself. As expected, the RMS error between prediction and fitting is, of course, smaller than the error between the prediction and SPICE. The difference

Signal	t_1	t_2	t_3	t_4	t_n
Input	(187.357, 10.335)	(-197.369, 10.677)	(186.633, 10.775)	(-199.268, 10.967)	...
Stage 1	(-199.655, 10.352)	(188.701, 10.695)	(-195.763, 10.792)	(180.644, 10.985)	...
Stage 2	(189.829, 10.370)	(-198.603, 10.712)	(184.386, 10.810)	(-210.301, 11.002)	...
Stage 3	(-198.686, 10.388)	(186.520, 10.730)	(-197.220, 10.827)	(186.032, 11.020)	...
Stage 4	(189.001, 10.406)	(-196.495, 10.747)	(187.225, 10.845)	(-199.534, 11.037)	...
Stage n	...	...	...	...	...

Table 5.1: (s_{i_n}, t_{i_n}) of inverter chain signals generated through fitting.

Signal	t_1	t_2	t_3	t_4	t_n
Input	(187.357, 10.335)	(-197.369, 10.677)	(186.633, 10.775)	(-199.268, 10.967)	...
Stage 1	(-198.383, 10.352)	(186.737, 10.694)	(-198.344, 10.792)	(186.935, 10.985)	...
Stage 2	(186.873, 10.370)	(-198.474, 10.712)	(188.413, 10.810)	(-199.010, 11.002)	...
Stage 3	(-198.435, 10.388)	(186.863, 10.730)	(-198.621, 10.828)	(186.949, 11.020)	...
Stage 4	(186.892, 10.406)	(-198.464, 10.747)	(188.372, 10.845)	(-199.005, 11.037)	...
Stage n	...	...	...	...	...

Table 5.2: (s_{i_n}, t_{i_n}) of inverter chain signals generated through prediction.

between the two error calculations is approximately equal to the fitting error.

Figure 5.3 depicts the error observed at each stage. The error compared to the fitting is approximately 0.5% after one stage, while after hundred stages it is about 3%. Interestingly, the error compared to SPICE is about 2.8% after one stage, while after hundred stages is it about 4.5%. One would have expected this error to be above the error for the fitting by a constant offset, no matter the stage.

5.3 Other Circuit Examples

Apart from the simple inverter chain, as elaborated in Section 5.2, we also investigated four other circuits. The first is an inverter chain consisting of `NOR` gates, where the input alternates with each stage. The second circuit is a `NOR` latch, which demonstrates the capability of our approach to handle short feedback loops. It will become apparent that the remarkable accuracy of the simple inverter chain, depicted in Figure 5.3, cannot be maintained for these other circuits. Additionally, we investigated the frequency (period) of `NOR` rings of different lengths in two different versions. Finally, we investigated how MIS effects affect the prediction performance in an `XOR` circuit composed of five `NOR` gates.

5.3.1 Inverter Chain with alternating Inputs

Figure 5.4 depicts an inverter chain, made from `NOR` gates, where the input tied to *GND* alternates between every stage. This is in contrast to the inverter chain that was

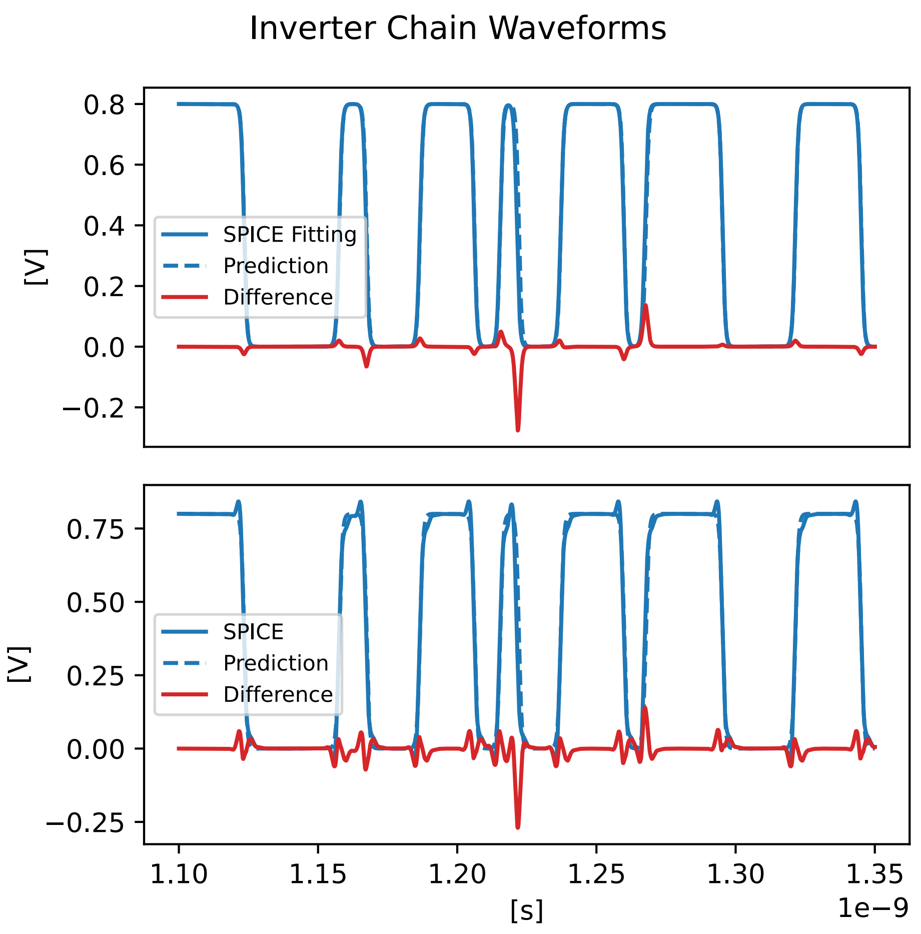


Figure 5.2: Inverter Chain Trace Comparison to Fitting and SPICE.

used for training data generation, see Figure 3.6: There, a chain of NOR gates was used, where either every gate in the chain had either its first input or its second input tied to *GND*. Consequently, no data was recorded for the "alternating" chain considered here. Of course, this case could have been covered by simulations and incorporated into the training set, leading to a more accurate prediction of this circuit. But we want to examine here how our approach deals with circuits it has never seen before.

The input stimulus and the number of simulations are the same as in the case of the conventional inverter chain studied in Section 5.2. Figure 5.5 depicts the measured error at each stage. As we can see, the error across the stages is substantially bigger than in Figure 5.3: After one stage, the error compared to the fitting is 3% and already 19% after ten stages, while the error to SPICE is 5% after one stage and 22% after ten stages.

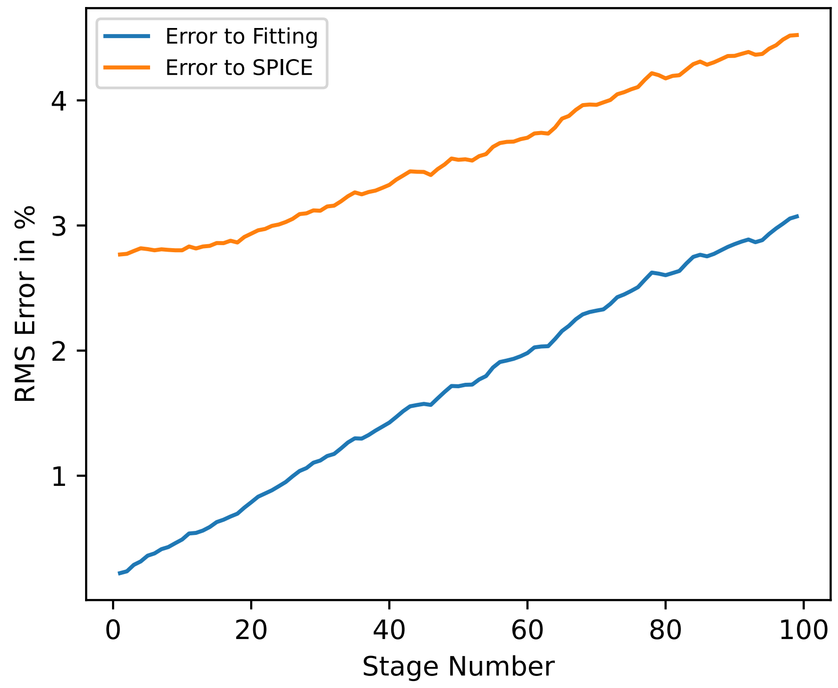


Figure 5.3: RMSE across an Inverter Chain.

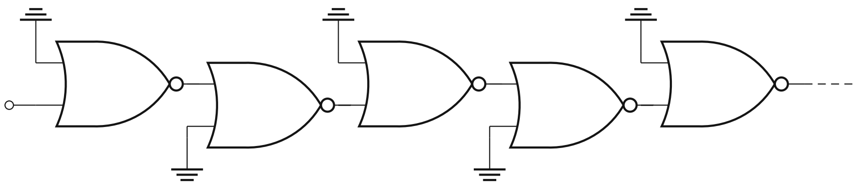


Figure 5.4: NOR inverter chain with alternating inputs.

5.3.2 NOR latch

The next circuit we investigated in our evaluation is a NOR latch. Figure 5.6 depicts the circuit with its inputs InA, InB and its outputs Q, $\bar{Q}$. Once again, the stimulus at the input will consist of random transition sequences with a Gaussian distribution of $\mu = 10\,\text{ps}$ and $\sigma = 5\,\text{ps}$.

Figure 5.7 depicts the example traces of the inputs and the predicted output of one of the 200 conducted runs. Even though the model has never been trained with simulation data that covers the behavior of a NOR gate where both inputs can switch, the results look very promising. Overall, we found that the prediction error of Q and $\bar{Q}$ is about 10.5% compared to the fitting and about 15% compared to SPICE. A closer look at Figure 5.7 shows that a lot of spikes appear in the SPICE waveform of Q that do not

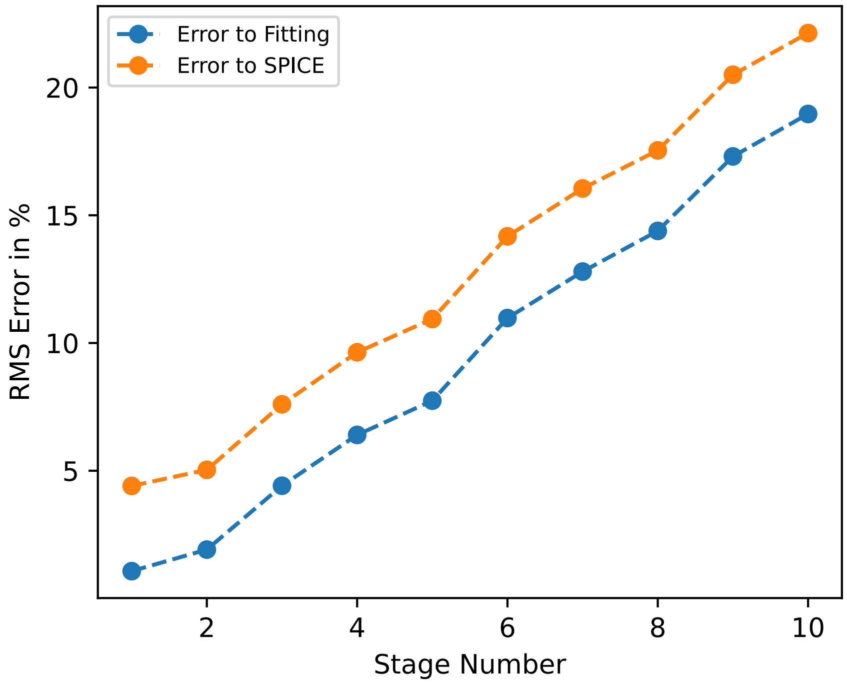


Figure 5.5: RMSE across a NOR inverter chain with alternating inputs.

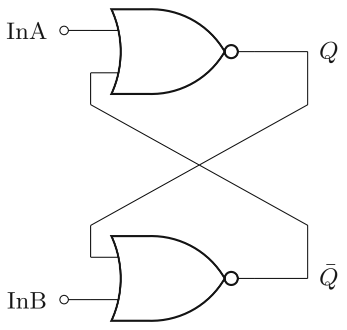


Figure 5.6: NOR Latch.

cross the threshold voltage, which might be the reason why the calculated error of Q is that high. Yet, the prediction matches the SPICE waveform very closely in the cases where the threshold is crossed. If we had defined our error metric in way that would focus only on threshold crossings (and would therefore be oblivious to sub-threshold spikes) we would definitely have calculated less than 15% error.

5.3.3 NOR rings

Another circuit we consider is an inverter ring composed of NOR gates. A “symmetric” version, as shown in Figure 4.1, and an “asymmetric” version, shown in Figure 5.8, are investigated. In the asymmetric version the NOR gate that is controlled by the input source gets its feedback through input B, whereas in the symmetric case all NOR gates use input A to form the ring. Through simulation in SPICE and the TOM, the period is calculated and compared for rings of different sizes. Table 5.3 lists the obtained periods for rings of sizes 3 to 11. As in the case of the inverter chains, a noticeable difference between the symmetric and asymmetric case can be observed. The prediction error in the case of symmetric rings is the highest in the case of 3 stages and settles to well below 1% for larger rings. This is in accordance with the errors that are in observed in a symmetric inverter chain, already presented in Figure 5.3. In the asymmetric case, rings with 3 and 5 stages also show promising prediction performance, albeit rings consisting of 7 stages and more show rather poor accuracy: In the case of 11 stages, an error of 17% is observed. Again, this is in accordance with the results obtained for inverter chains with alternating inputs, see Figure 5.5. This once again highlights the fragility of ANN-based predictions when analyzing scenarios that are not in the training set.

Circuit	SPICE period[ps]	predicted period[ps]	Difference
3 stages symmetric	15.867	15.453	-2.613%
5 stages symmetric	27.378	27.323	-0.202%
7 stages symmetric	38.448	38.515	0.175%
9 stages symmetric	49.475	49.426	-0.099%
11 stages symmetric	60.504	60.410	-0.155%
3 stages asymmetric	15.233	14.994	-1.569%
5 stages asymmetric	26.817	26.705	-0.421%
7 stages asymmetric	37.887	34.757	-8.262%
9 stages asymmetric	48.909	41.792	-14.551%
11 stages asymmetric	59.932	49.721	-17.039%

Table 5.3: Comparison of the period in NOR rings of different sizes.

5.3.4 XOR gate

Finally, we investigate a small combinatorial circuit where both inputs of a gate can switch simultaneously. Figure 5.9 depicts an XOR gate made of NOR gates. The two left most NOR gates serve as inverters and do not exhibit any multi-input switching (MIS) effects, but the inputs of the remaining three NOR gates may switch at the same time. Since MIS effects, which can speed-up or slow down the signal propagation through a NOR gate [13], are not considered in the TOM, a limited prediction performance is to be expected.

As in the earlier examples, the inputs of the circuit were stimulated with random transition sequences. Figure 5.10 shows an example waveform and the corresponding transition of the circuit output. The prediction error over 200 runs was found to be 8% compared to the fitting and 10.5%. It appears that the lacking MIS effect modeling capabilities of our TOM had a low impact on the prediction performance. Again, judging by Figure 5.10, a main source of error seems to be the over- and undershoots at the start of a transition, which are generally uninteresting with regards to timing analysis.

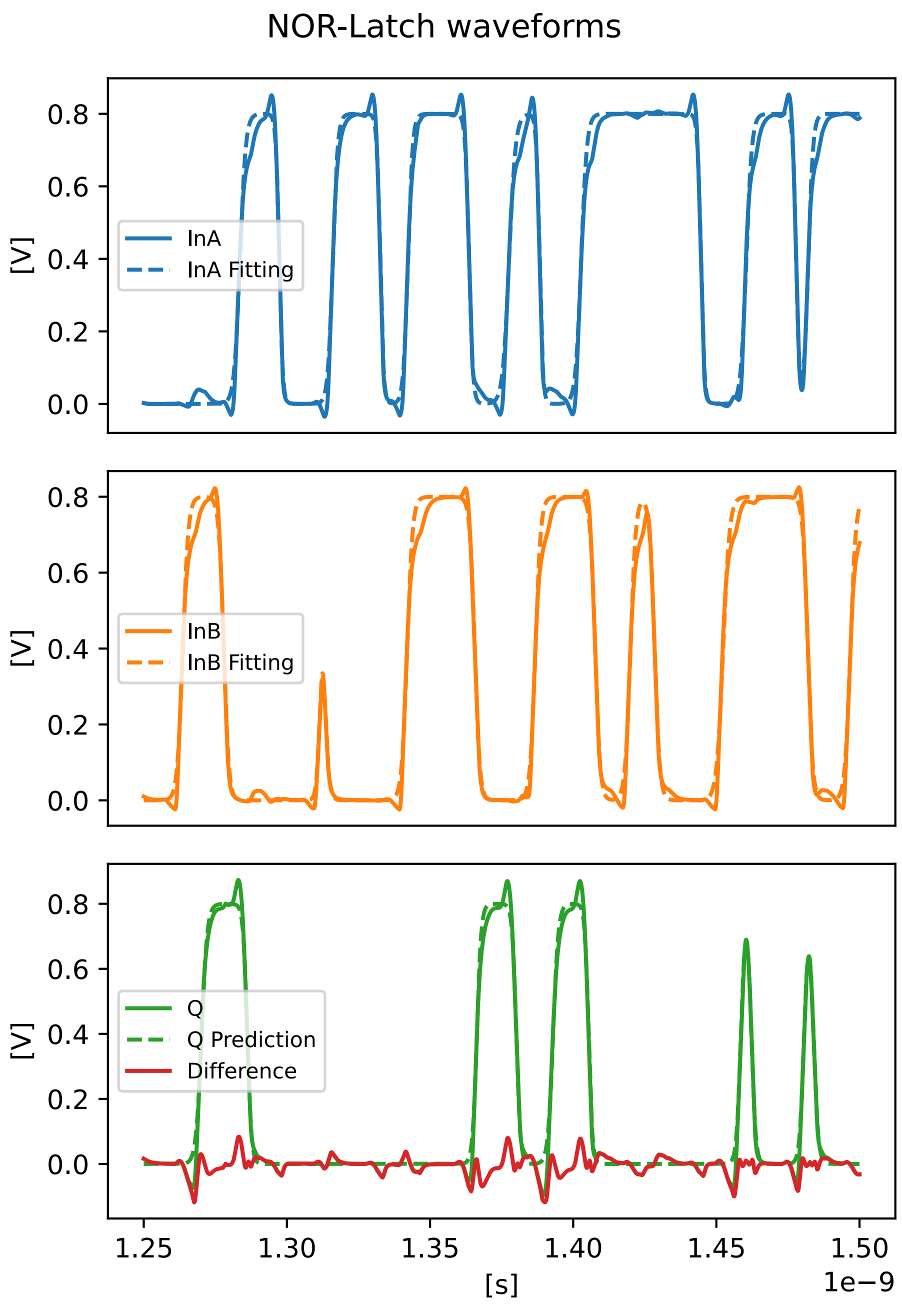


Figure 5.7: Example Traces of Inputs and Output of the NOR Latch.

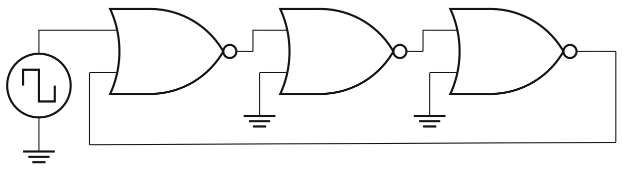


Figure 5.8: Three NOR gates in a ring, feedback through input B.

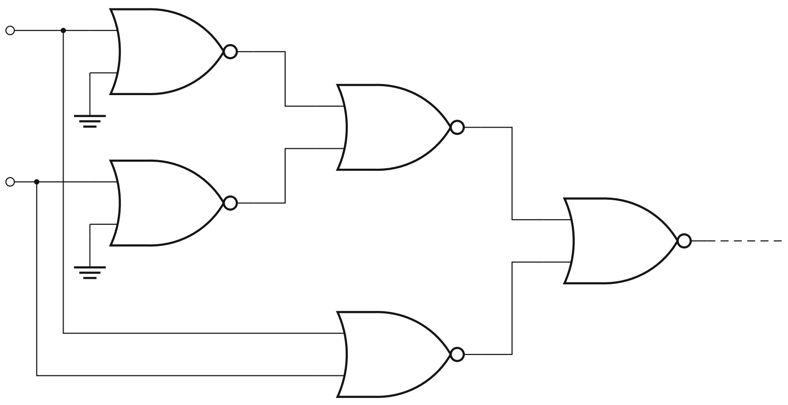


Figure 5.9: XOR gate built from NOR gates

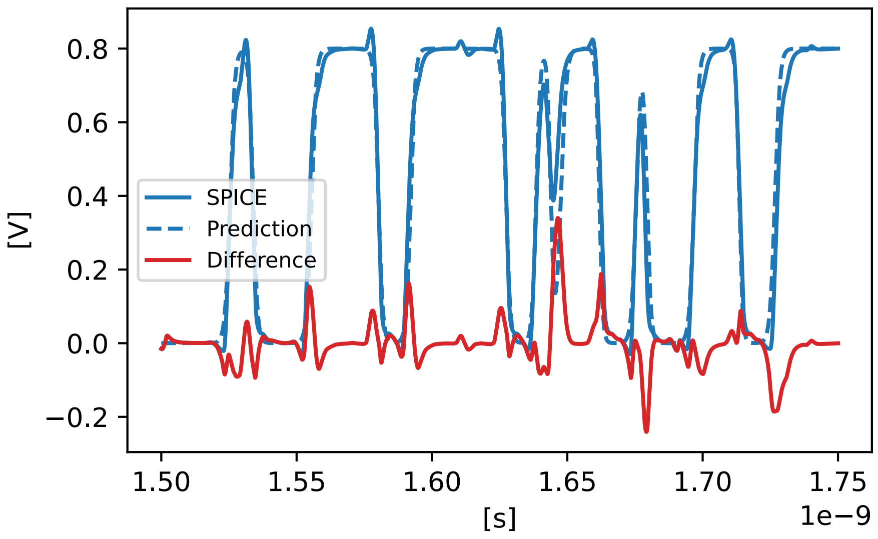


Figure 5.10: Example trace at the output of the XOR gate

CHAPTER 6

Shortcomings and Remarks

In this chapter, we present some shortcomings of the proposed approach, which could not be addressed in this thesis within the available time frame.

6.1 General Output Loads

One of the main unaddressed aspects of of the proposed approach is how to handle varying output loads. A realistic model would somehow have to incorporate this information into its prediction. One possible approach would be to develop a theory of sigmoidal transfers across RC-circuits, similar to Elmore delays [35], which approximate the effects an RC-circuit imposes on sigmoids. Another straightforward way would be to subdivide the domain of possible output capacitances, similar to CCS [5], and compute transfer relations for each section. It should also be noted here that we did not consider process voltage and temperature (PVT) variations, which considerably affect the operation of digital circuits [36].

6.2 Intractable source of error

Although we deliberately restricted our attention to gates that are connected to exactly one other gate of the same kind in this thesis, we also stumbled over an interesting effect in the case of a fan-out larger than one: It appears that the voltage of one input of a `NOR` gate affects its other input. To illustrate this effect, consider the circuit in Figure 6.1, where a `NOR` gate with fan-out of four is depicted in three different scenarios.

In case A, all four output gates have their other input connected to *GND*, in case B two inputs are connected to *GND* and two to V_{DD}, while in case C all other inputs are connected to V_{DD}. Of course, the parasitics of the circuit were made symmetric in the case of the connection between G_1 and G_{2-5}. We used the parasitic network depicted in Figure 6.3, which is a slightly modified version of a Cadence-generated one. All parasitic

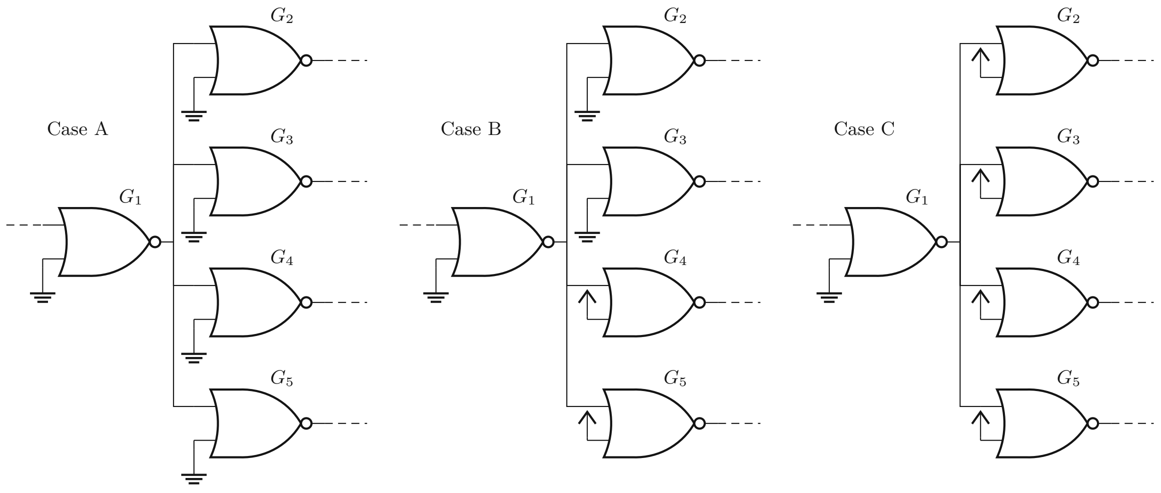


Figure 6.1: NOR gate with fan-out of four in different scenarios.

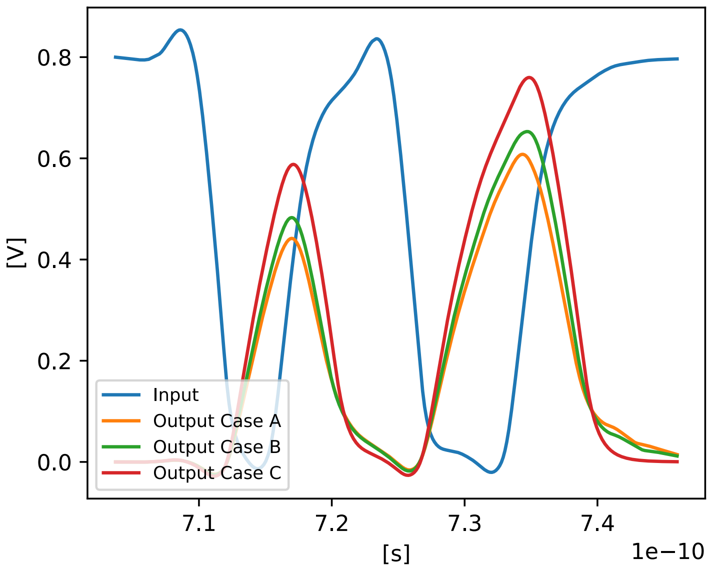


Figure 6.2: Input and Output Waveforms in all scenarios.

networks between the input and G_1 and between G_{2-5} and their output are the one shown in Figure 3.2.

When stimulating the input of the NOR gate G_1 with the same waveform in every scenario, we observed that its output waveform considerably varies. Figure 6.2 shows an example input waveform with the three different output waveforms for each case. We can observe that in case A the output steepness is the smallest, while in case C it is the highest. Apparently, the output steepness gets higher as more inputs are set to

Signal	t_1	t_2	t_3	t_4
Input	(-186.41,7.11)	(149.00, 7.17)	(-185.92, 7.25)	(105.42, 7.35)
Output Case A	(97.22, 7.15)	(-68.03, 7.18)	(62.15, 7.30)	(-65.68, 7.36)
Output Case B	(97.40, 7.15)	(-74.65, 7.18)	(61.21, 7.30)	(-78.14, 7.37)
Output Case C	(94.03, 7.14)	(-97.51, 7.19)	(72.06, 7.29)	(-104.91, 7.37)

Table 6.1: Fitting Parameters of Figure 6.2.

V_{DD}. Note that, in the case of the other input being V_{DD}, the input transition does not propagate. A naive interpretation of this phenomenon may be that transitions that do not trigger output transitions require no work from the receiving gate to switch its output, resulting in a lower input capacitance of the respective input.

Also note that the three scenarios are such that the unused inputs are supplied with a constant voltage. Cases where the other inputs also switch where not considered.

Regardless of the underlying physical phenomenon governing this behavior, it imposes a considerable accuracy penalty to our modeling approach. Table 6.1 lists the fitting results of the waveforms shown in Figure 6.2. Considering that the fitting parameters for the input are the same in all three cases, the difference in fitting parameters between the three cases is remarkable. The steepness of the falling transitions varies by more than 30% from case A to case C. Since it is highly unreasonable to incorporate the input states of the successor gate(s) into the transition computation for a gate, there is no chance of keeping track of this behavior when modeling such circuits. It is unclear how to construct a reasonable model covering this behaviour.

6.3 Future Work

Although a proof of concept has been provided in this thesis, we could not investigate every aspect of our approach in depth. Several questions remain unanswered: What is the optimal sigmoidal function for fitting analog waveforms? What is the optimal ANN topology for predicting fitting parameters? How to deal with arbitrary interconnect or gates that have arbitrary fan-out? How to model process voltage and temperature (PVT) variations? We believe that satisfying answers to these questions would yield a simulation tool competitive to existing ones.

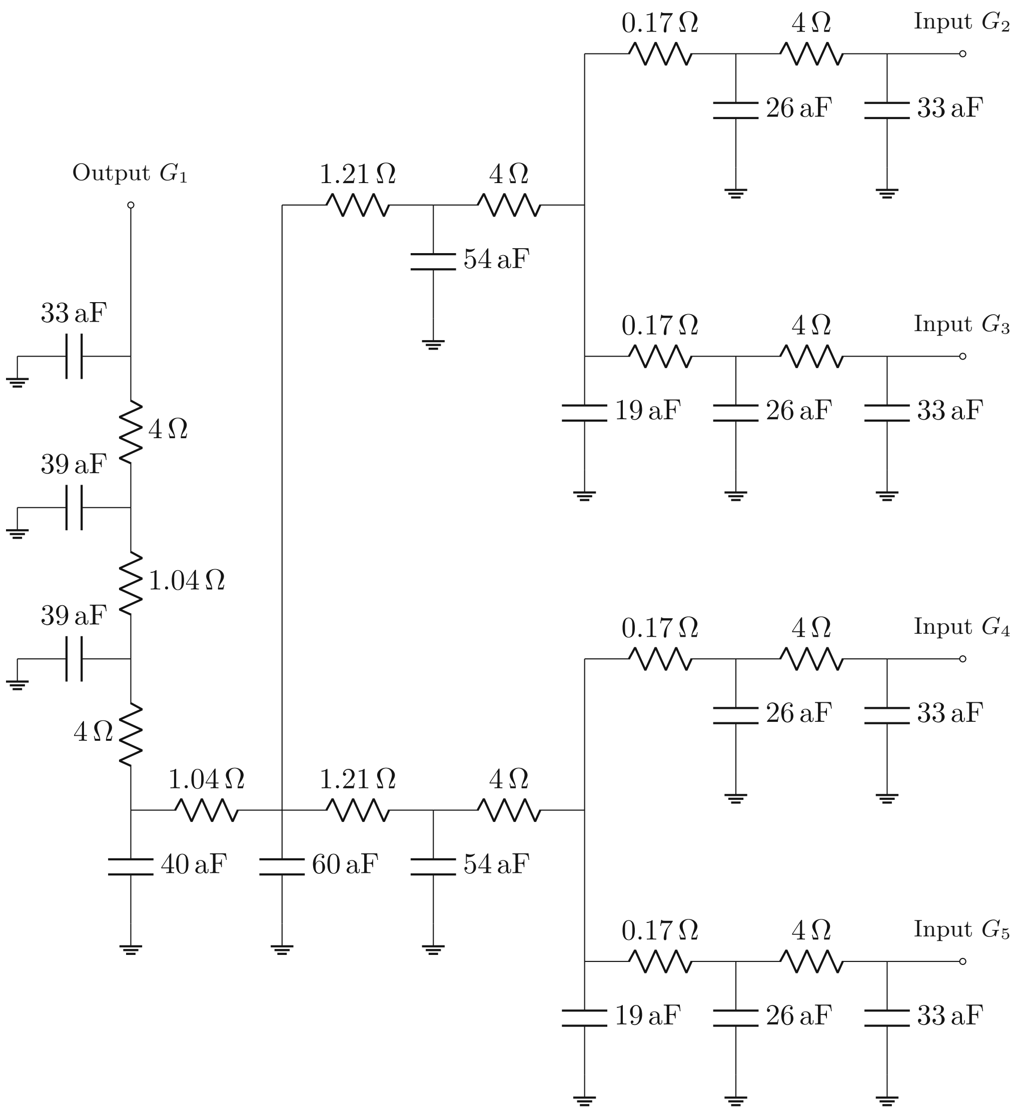


Figure 6.3: Circuit used as Parasitic Network for the case of Fan-out=4.

CHAPTER 7

Conclusions

This thesis presented a novel approach for simulating the temporal behavior of digital circuits, which complements existing approaches by combining the advantages of both analog and digital simulators. It is based on approximating voltage waveforms of digital circuits with parameterized sigmoids and using the list parameters representing a waveform for predicting the output waveforms of inverters and `NOR` gates. We presented the core ideas and the means of characterizing logic gates using the example of 15 nm technology inverters and `NOR` gates. The approach easily translates to other technologies and logic gates, however. When comparing the predictions of our approach against SPICE, we observed errors in the range of a few percent, while spending considerably less time for simulating. As the implementation of our approach has not been exposed to several years of refinement, we do believe that even better simulation-time performance could be achieved.

List of Figures

List of Tables

Bibliography

[1] Laurence Nagel and Donald O Pederson. Spice (simulation program with integrated circuit emphasis). 1973.

[2] B. J. Sheu, D. L. Scharfetter, P. . Ko, and M. . Jeng. Bsim: Berkeley short-channel igfet model for mos transistors. *IEEE Journal of Solid-State Circuits*, 22(4):558–566, Aug 1987.

[3] Y. S. Chauhan, S. Venugopalan, N. Paydavosi, P. Kushwaha, S. Jandhyala, J. P. Duarte, S. Agnihotri, C. Yadav, H. Agarwal, A. Niknejad, and C. C. Hu. Bsim compact mosfet models for spice simulation. In *Proceedings of the 20th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2013*, pages 23–28, 2013.

[4] Effective Current Source Model. Timing and power specification. *Cadence Design Systems*, 2015.

[5] CCS Timing Library Characterization Guidelines. *Synopsis Inc.*, October 2016, version 3.4.

[6] S. H. Unger. *Asynchronous sequential switching circuits.* Wiley-Interscience, 1969.

[7] M. J. Bellido-Diaz, J. Juan-Chico, A. J. Acosta, M. Valencia, and J. L. Huertas. Logical modelling of delay degradation effect in static CMOS gates. *IEE Proceedings - Circuits, Devices and Systems*, 147(2):107–117, April 2000.

[8] Manuel J Bellido, Jorge Juan Chico, and Manuel Valencia. *Logic-timing simulation and the degradation delay model.* Imperial College Press, 2005.

[9] Matthias Függer, Robert Najvirt, Thomas Nowak, and Ulrich Schmid. A faithful binary circuit model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2784–2797, 2019.

[10] Daniel Öhlinger. Involution tool. *E191-Institut für Computer Engineering*, 2018.

[11] J.C. Ebergen, S. Fairbanks, and I.E. Sutherland. Predicting performance of micropipelines using charlie diagrams. In *Proceedings Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 238–246, 1998.

[12] Daniel Öhlinger, Jürgen Maier, Matthias Függer, and Ulrich Schmid. The involution tool for accurate digital timing and power analysis. *Integration*, 76:87–98, 2021.

[13] Arman Ferdowsi, Jürgen Maier, Daniel Öhlinger, and Ulrich Schmid. A simple hybrid model for accurate delay modeling of a multi-input gate. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1461–1466. IEEE, 2022.

[14] Marcus Ossiander, Keyhan Golyari, K Scharl, Lukas Lehnert, Florian Siegrist, JP Bürger, Dmitry Zimin, JA Gessner, Matthew Weidman, I Floss, et al. The speed limit of optoelectronics. *Nature communications*, 13(1):1–8, 2022.

[15] Yuanping Song, Robert M Panas, Samira Chizari, Lucas A Shaw, Julie A Jackson, Jonathan B Hopkins, and Andrew J Pascall. Additively manufacturable micro-mechanical logic gates. *Nature communications*, 10(1):1–6, 2019.

[16] Eran A Barnoy, Menachem Motiei, Chen Tzror, Shai Rahimipour, Rachela Popovtzer, and Dror Fixler. Biological logic gate using gold nanoparticles and fluorescence lifetime imaging microscopy. *ACS Applied Nano Materials*, 2(10):6527–6536, 2019.

[17] R Jacob Baker. *CMOS: circuit design, layout, and simulation*. John Wiley & Sons, 2019.

[18] `https://github.com/JosefSalzmann/Sigmoidal_Waveform_Approximation`.

[19] Eric Schneider, Stefan Holst, Xiaoqing Wen, and Hans-Joachim Wunderlich. Data-parallel simulation for fast and accurate timing validation of CMOS circuits. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 17–23. IEEE, 2014.

[20] O. V. S. Shashank Ram and Sneh Saurabh. Modeling multiple-input switching in timing analysis using machine learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(4):723–734, 2021.

[21] M Gunes, MA Thornton, Fatih Kocan, and SA Szygenda. A survey and comparison of digital logic simulators. In *48th Midwest Symposium on Circuits and Systems, 2005.*, pages 744–749. IEEE, 2005.

[22] Peter Roessler, Roland Höller, Christopher Reisner, and Oliver Maischberger. Survey and comparison of digital logic simulators. In *2019 Austrochip Workshop on Microelectronics (Austrochip)*, pages 87–92. IEEE, 2019.

[23] V Chandramouli and Karem A Sakallah. Modeling the effects of temporal proximity of input transitions on gate propagation delay and transition time. In *Proceedings of the 33rd annual Design Automation Conference*, pages 617–622, 1996.

[24] José Mira and Francisco Sandoval. *From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks, Malaga-Torremolinos, Spain, June 7-9, 1995: Proceedings*, volume 930. Springer Science & Business Media, 1995.

[25] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[26] Josef Salzmann. Approximating analog waveforms by adding arbitrary functions. B.S. Thesis, TU Wien, 2021.

[27] Mayler Martins, Jody Maick Matos, Renato P Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. Open cell library in 15nm freepdk technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, pages 171–178, 2015.

[28] Ake Bjorck. *Numerical methods for least squares problems*, volume 51. Siam, 1996.

[29] Lowell Jacob Reed and Joseph Berkson. The application of the logistic function to experimental data. *The Journal of Physical Chemistry*, 33(5):760–779, 2002.

[30] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[31] Adriano Moreira and Maribel Yasmina Santos. Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. 2007.

[32] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE, 2011.

[33] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. On the design of cgal a computational geometry algorithms library. *Software: Practice and Experience*, 30(11):1167–1202, 2000.

[34] Peter B Andrews. *An introduction to mathematical logic and type theory: to truth through proof*, volume 27. Springer Science & Business Media, 2013.

[35] R. Gupta, B. Tutuianu, and L.T. Pileggi. The elmore delay as a bound for rc trees with generalized input signals. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(1):95–104, 1997.

[36] Jayaram Bhasker and Rakesh Chadha. *Static timing analysis for nanometer designs: A practical approach*. Springer Science & Business Media, 2009.