Involution Tool

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

Daniel Öhlinger
Matrikelnummer 01525898

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Ulrich Schmid
Mitwirkung: Univ. Ass. Dipl.-Ing. Dipl.-Ing. Jürgen Maier, BSc

Wien, 19. Dezember 2018

Daniel Öhlinger	Ulrich Schmid

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Involution Tool

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

Daniel Öhlinger
Registration Number 01525898

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Ulrich Schmid
Assistance: Univ. Ass. Dipl.-Ing. Dipl.-Ing. Jürgen Maier, BSc

Vienna, 19th December, 2018

Daniel Öhlinger	Ulrich Schmid

Erklärung zur Verfassung der Arbeit

Daniel Öhlinger
Koppstraße 68/39
1160 Wien

Wien, 19. Dezember 2018

Daniel Öhlinger

Danksagung

Zuallererst möchte ich mich bei Prof. Schmid bedanken, der mir die Möglichkeit gegeben hat, meine Bachelorarbeit in einem aktiven Forschungsfeld zu schreiben. Die Arbeit wurde im Kontext des FWF-Projekts SIC (P26436) durchgeführt und mit einem Bachelor with Honors Scholarship (Forschungsbeihilfe) gefördert. Es war sehr spannend für mich, ein Tool zu schreiben welches tatsächlich benützt werden kann, um aktiv zur Forschung beizutragen.

Außerdem möchte ich mich bei Jürgen Maier bedanken, der mir immer mit Rat und Tat zur Seite stand, wenn es während der Entwicklung Fragen gab. Seine Antworten waren immer hilfreich und unglaublich schnell.

Mein besonderer Dank gilt meinen Eltern, die mich in all meinem Entscheidungen bestärkt haben.

Ein großer Dank gilt auch Sophia, die mich während des gesamten Studiums unterstützt hat, und auch akzeptiert hat, wenn ich vor einer Abgabe wieder einmal tagelang im Labor war.

Zu guter Letzt möchte ich mich bei Clemens und Wolfgang bedanken, mit denen ich gemeinsam das Bachelorstudium bestritten habe. Dank euch war das Studium nicht nur unglaublich interessant, sondern auch sehr unterhaltsam.

Acknowledgements

First and foremost I would like to thank Prof. Schmid for giving me the opportunity to write my Bachelor's thesis in an active field of research. My work was conducted in the context of the FWF project SIC (P26436), and supported by a Bachelor with Honors FWF scholarship (Forschungsbeihilfe). It was really exciting to write a tool which can be actively used to contribute to research.

Moreover, I would like to thank Jürgen Maier, who was always ready to help when there had been questions during the development of the tool. His advice was always helpful and incredibly fast.

Further, I would like to extend my gratitude to my parents who reinforced me in all my decisions.

A big thank you goes out to Sophia, who supported me throughout my studies, and also accepted it when I spent my whole time in the laboratory before hand-ins.

Finally, I would like to thank Clemens and Wolfgang, with whom I had contested my Bachelor studies. Thanks to you my studies were not only incredibly interesting, but also very enjoyable.

Kurzfassung

Das Zeitverhalten komplexer digitaler Schaltungen ist sehr schwierig vorherzusagen. Besonders für große Schaltungen ist die Simulation auf Basis analoger Modelle zu langsam. Daher ist eine digitale Abstraktion erforderlich, um die praktische Durchführbarkeit solcher Simulationen zu gewährleisten.

Eine mögliche Grundlage für eine digitale Abstraktion ist das Involution Modell, welches im Gegensatz zu anderen Verzögerungsmodellen die Wirklichkeit originalgetreu abbildet. Ein Modell wird originalgetreu genannt, wenn es alle Probleme die in einer echten Schaltung gelöst werden können auch lösen kann, und umgekehrt. Allerdings existiert das Involution Modell hauptsächlich in der Theorie, daher ist es das Ziel dieser Bachelorarbeit, ein Tool zur Verfügung zu stellen, welches dieses Modell verwendet. Hierbei wird das Modell in der weitverbreiteten Simulationsplattform ModelSim implementiert, mit welcher dann Zeitvorhersagen gemacht werden können. Das Tool vergleicht außerdem die Resultate mit anderen Modellen und analogen Simulationen.

Der Vergleich des Involution Modells mit dem in ModelSim integrierten Standard-Modell zeigte signifikante Performanceverbesserung in Bezug auf die Leistungsschätzung und der Genauigkeit bei der Modellierung der Signale. Für eine Schaltung aus Invertern konnte die Abweichung bezüglich der Leistungsschätzung von ungefähr 53% auf 15% reduziert werden.

Abstract

The timing behavior of complex digital circuits is very hard to predict. Using analog models for simulation is, especially for larger circuits, too slow. Therefore, a digital abstraction is required in order to ensure feasible simulations.

A promising basis for such a digital abstraction is the involution model, which differs from other delay models in that it is faithful. A model is faithful if problems, which can be solved with a real physical circuit, can be solved in the model and vice versa. Since the involution model mainly exists on paper, however, the goal of this Bachelor's thesis is to provide a tool which makes the model applicable. The tool implements the delay model in the popular simulation platform ModelSim, which facilitates timing predictions of arbitrary circuit designs. It also compares the results to other models and analog simulations.

The comparison of the involution model with the built-in delay method showed significant performance improvements in terms of power estimation and trace modeling accuracy. For a circuit consisting of inverters, the power estimation error could be decreased from approximately 53% to 15%.

Contents

CHAPTER 1

Introduction

Digital circuits nowadays play a vital role in every part of our everyday life. The high demand for improved performance and functionality leads to an over increasing complexity of these circuits. As consequence the effort to check correct functionality, i.e., that the circuit delivers the correct signal at the right time, also increases. One possibility are fully-fledged *analog simulations*, which model the analog signal (continuous in time and value), using very accurate but also rather complex differential equations. Thus, they are only applicable for small circuits. Therefore, one has to resort to digital delay models (continuous in time, discrete in value), which facilitate very fast timing simulations.

Delay models try to model the behavior of signals in a digital circuit and are used in timing simulation tools and also for correctness proofs. It is obvious that they must accurately model the physical reality in order to be meaningful. Surprisingly, this is not the case for any existing binary-valued circuit model, as proved in [FNS13]. The authors showed that all the existing binary delay models do not faithfully model the propagation of short pulses, called glitches, in digital circuits. A model is unfaithful if it is either not possible to solve a problem in the model that can be solved with a physical circuit or vice versa.

[FNN15] came up with a delay model that faithfully solves the problem of the propagation of short pulses. This model, called *involution delay model*, is a promising candidate for replacing existing delay models in simulation tools and for future formal verification of digital circuits.

A lot of theoretical research has already been made on the involution delay model. Also simulations and measurements on hardware have been carried out, but the involution delay model still lacks tool support. Other delay models, despite the drawback of being unfaithful, are heavily used in simulation tools like ModelSim, NC-Sim and VCS. Hence, the goal of this Bachelor's thesis is to provide a tool with which the involution delay model can be compared to analog SPICE simulations, and other existing (unfaithful) delay models. One main contribution is the possibility to simulate the involution delay model with ModelSim and automatically compare the results to the built-in method. All the results from different simulations are aggregated and can be exported or displayed in a report.

The structure of the thesis is as follows: State of the art delay models are introduced in Chapter 2. Their differences are discussed, and it is explained why all of them are unfaithful. In this chapter we also explain the involution model. In Chapter 3 we explain the different parts of the toolchain and how these parts work together. We also justify design decisions that have been made during the implementation and discuss open problems. Chapter 4 focuses on the simulation of circuits, using our Involution Tool. Two test circuits have been employed, one consisting of inverters and the other one consisting of NAND gates. Especially the first test circuit resulted in good results. Chapter 5 summarizes the accomplishments, discusses open problems and gives an outlook for possible future work. In Appendix A, the manual for the Involution Tool can be found, Appendix B provides the VHDL implementation of the involution channel.

CHAPTER 2

State of the Art

2.1 Delay models

Delay models are used for timing analysis of digital circuits and are employed in state-of-the-art tools like ModelSim. They try to accurately model the propagation of transitions from the input of a gate to the output.

A very simple delay model is the so-called *pure delay model.* It adds a constant delay to the signal for the transmission between input and output. A more advanced approach is the *inertial delay model* [Ung70]. This delay model blocks pulses with a width $\leq \Delta$; If there is a subsequent opposite transition in a time window Δ after the current transition, the pulse is canceled out. This means that a short pulse on the input is not propagated to the output.

A big flaw of the inertial delay model is the discontinuity between the inertial and the propagation region. Pulses with a width $\leq \Delta$ are canceled out, whereas longer pulses $\Delta + \delta$, with δ arbitrarily small, are propagated without modification. Real physical gates do not have this discontinuity. Therefore, the *Degradation Delay Model* (DDM), a more accurate model, has been introduced by Bellido-Diaz et al. [BJA00], [BJV05]. It introduces a so-called degradation region between the inertial and propagation region. If an input transition arises, but the gate has not fully switched yet, the degradation effect takes place, which leads to a shorter pulse on the output. The shorter the pulse on the input, the more the pulse width on the output is reduced.

DDM is a so-called *bounded single-history channel*, which is a generalization of constant delay models. Each input event, occurring at time t, is propagated to the output after a bounded delay $\delta(T)$. The parameter $T = t - t'$ is the input-to-previous-output delay, where t' is the time of the last output event. Note that $\delta(T)$ can be negative, and the most negative delay, as well as the most positive delay, is bounded.

Figure 2.1 shows a comparison of the presented delay models. We can see that the pure delay passes the transitions to the output with a constant delay. The inertial delay model does not propagate (2) because the pulse length is $\leq \Delta$. For DDM we can see the three different regions. (1) is long enough to be in the pure delay region and therefore propagated without modification. (2) is in the inertial region and therefore canceled out. The degradation effect can be seen at (3), where the pulse length at the output is reduced, compared to the input.

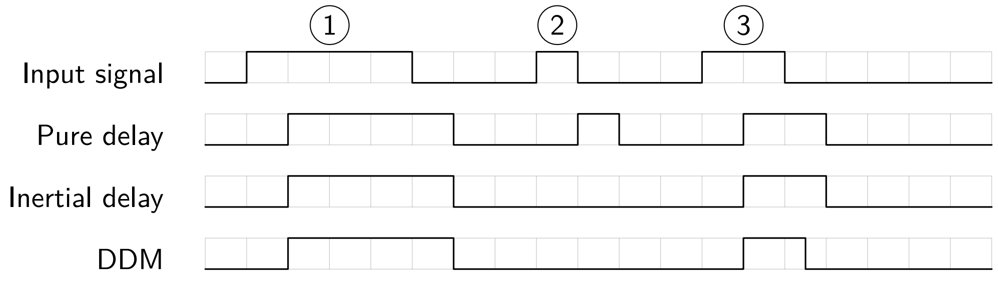

Figure 2.1: Comparison of different delay models.

Tools like Synopsis VCS and Cadence NC-Sim use sophisticated techniques to characterize, i.e., determine actual gate delays. Synopsis VCS uses a library called *Composite Current Source* (CCS) [Syn16], whereas Cadence NC-Sim uses the *Effective Current Source Model* (ECSM) [Cad15] library. They perform dynamic simulations and consider different technology dependent parameters like driving strength and load capacities. The results of these simulations are then used to compute accurate bounds for parametrizing the delay channels. However, these delays are pre-computed and do not change during subsequent simulation runs.

2.2 Faithful delay models

Faithful delay models are of utter importance when it comes to (i) fast digital timing simulation and especially (ii) correctness proofs. A delay model which is fast, sufficiently accurate and also realistic is called *faithful*. Realisitic means that a problem is solvable in the model if and only if it is solvable by a physical circuit. In [FNS13] and [FNS16] it has been shown that all three aforementioned delay models are not realistic, since they either contradict the unsolvability of the *Short-Pulse Filtration* (SPF) problem in bounded time, or the solvability of SPF in unbounded time.

One can think of the SPF problem as the problem of building a one shot version of an inertial delay channel. Short pulses with a length of $\epsilon > 0$ may not appear on the output, while longer pulses are passed unaltered or extended. The bounded version of SPF requires the output to make its decision in a bounded time $T > 0$, which means that after the time $t + T$, where t is the time of the last input transition, the output is stable and does not change any more.

In [FNS13] and [FNS16] it is shown that pure delay models cannot solve the SPF problem, even in unbounded time, which contradicts the physical reality. In the same papers, it is also shown that with non-constant bounded delay channels (like DDM and inertial delay model) bounded SPF is solvable, which is again a contradiction to the physical reality, as proved in [Mar77].

2.3 Involution model

Since all the previously explained delay models suffer from the deficiency of unfaithfulness, the *involution model* has been introduced in [FNN15], experimentally simulated in [NSH15] and further extended in [FMN18]. It is a promising candidate for more accurate simulation tools and formal verification, since it realistically models glitch propagation.

The involution delay model has, in sharp contrast to the previously presented delay models, *unbounded* delay functions: The delay is still upper bounded, but not bounded from below. One key property of the involution channels is that the negative delay functions have to be involutions: $-\delta_\downarrow(-\delta_\uparrow(T)) = T$ and $-\delta_\uparrow(-\delta_\downarrow(T)) = T$. Note that the delay functions for rising transitions $\delta_\uparrow$ and falling transitions $\delta_\downarrow$ can be different. Figure 2.2 shows a typical involution delay function with different pure delays, where the parameter T is again the input-to-previous-output delay. In Figure 2.3 one can see how the the parameter T (T_1, T_2) is derived and the corresponding delay $\delta(T)$ $(\delta_\uparrow(T_1), \delta_\downarrow(T_2))$ is calculated.

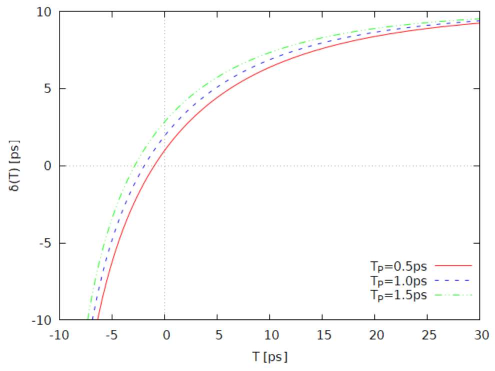

Figure 2.2: Involution delay functions with different T_P.

In [FNN15], a simple analog channel model that justifies the use of involution delay functions is given. This analog channel model can be seen in Figure 2.3. It consists of a pure delay T_P, a slew rate limiter that converts the step functions (u_d) to instances of $f_\uparrow$ and $f_\downarrow$, and a comparator that discretizes the continuous waveform again (u_o). When considering the special case that the slew rate limiter is implemented as a first-order RC low pass filter, a certain involution channel type, called *exp-channel*, is received. The tool implemented during this Bachelor's thesis uses precisely this type of involution channels with the following delay functions (where τ_{up} and τ_{do} are the RC-constants):

$$\delta_\uparrow(T) = \tau_{up} \ln\left(1 - e^{-\frac{T+T_P-\tau_{do}\ln\left(\frac{V_{th}}{V_{dd}}\right)}{\tau_{do}}}\right) + T_P - \tau_{up}\ln\left(1 - \frac{V_{th}}{V_{dd}}\right)$$

$$\delta_\downarrow(T) = \tau_{do} \ln\left(1 - e^{-\frac{T+T_P-\tau_{up}\ln\left(1-\frac{V_{th}}{V_{dd}}\right)}{\tau_{up}}}\right) + T_P - \tau_{do}\ln\left(\frac{V_{th}}{V_{dd}}\right)$$

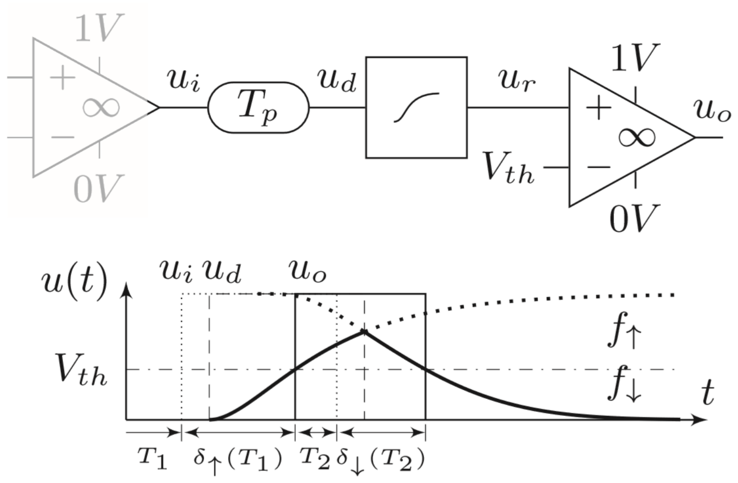

Figure 2.3: Simple analog channel model from [FNN15].

CHAPTER 3

Implementation - Involution Tool

The *Involution Tool* is based on several parts, which are going to be described in the following sections. The main focus was to provide a tool where simulating new circuits is easy, but also keeping the tool flexible enough for circuit specific requirements. It only takes a few steps from adding a new circuit to receiving simulation results. Each step of the toolchain can be replaced by an own implementation, as long as the interface to the previous and following step stays the same. This makes it easy to implement custom optimizations. For example, exchanging the tool ModelSim with another simulation tool should only affect the simulation step of the toolchain, but not the other parts.

The Involution Tool consists of the following parts, which are also shown in Figure 3.1:

- Cadence Encounter: Before starting to evaluate a circuit we first have to design the circuit and make the layout. During this step we create a Verilog (*.v) file and a SPICE (*.sp) file describing the circuit, and also information about the circuit delays (*.sdf). One could of course use a different tool than Cadence Encounter. The important thing is to get the three aforementioned files, which are used at various steps of the toolchain.
- Waveform generation (Section 3.1): The first step of the tool is to generate the input stimuli which is applied to the circuit. It uses information about the circuit (*.v) and fills the SPICE template (*.sp).
- HSPICE (Section 3.2): In this step the circuit is simulated with the previously generated input stimuli. The output is a trace file (*.tr0) containing the traces of all specified signals and a measurement file (*.mt0) containing for example information about the power consumption.
- ModelSim (Section 3.3): Before the traces (*.tr0) can be applied to a ModelSim simulation we need to digitize them (Crossings / Vectors). The result (vectors_*) is then applied via testbench.

- Power estimation (Section 3.4): For estimating the power the output of the ModelSim simulation (*.vcd) is used and either converted into an intermediate format (*.saif) or directly applied to different power estimation tools.
- Evaluation (Section 3.5): In this step the results of all previous steps are accumulated and converted into a common data representation (results.json). The tool also does some preparations for the reporting, so that we do not have to parse the data again in the reporting step.
- Reporting (Section 3.6): Generates a LaTeX report from the information that has been extracted in the previous step.

The following Sections 3.1-3.7 aim to explain design decisions made during the implementation. A manual for the toolchain with configuration examples can be found in Appendix A.

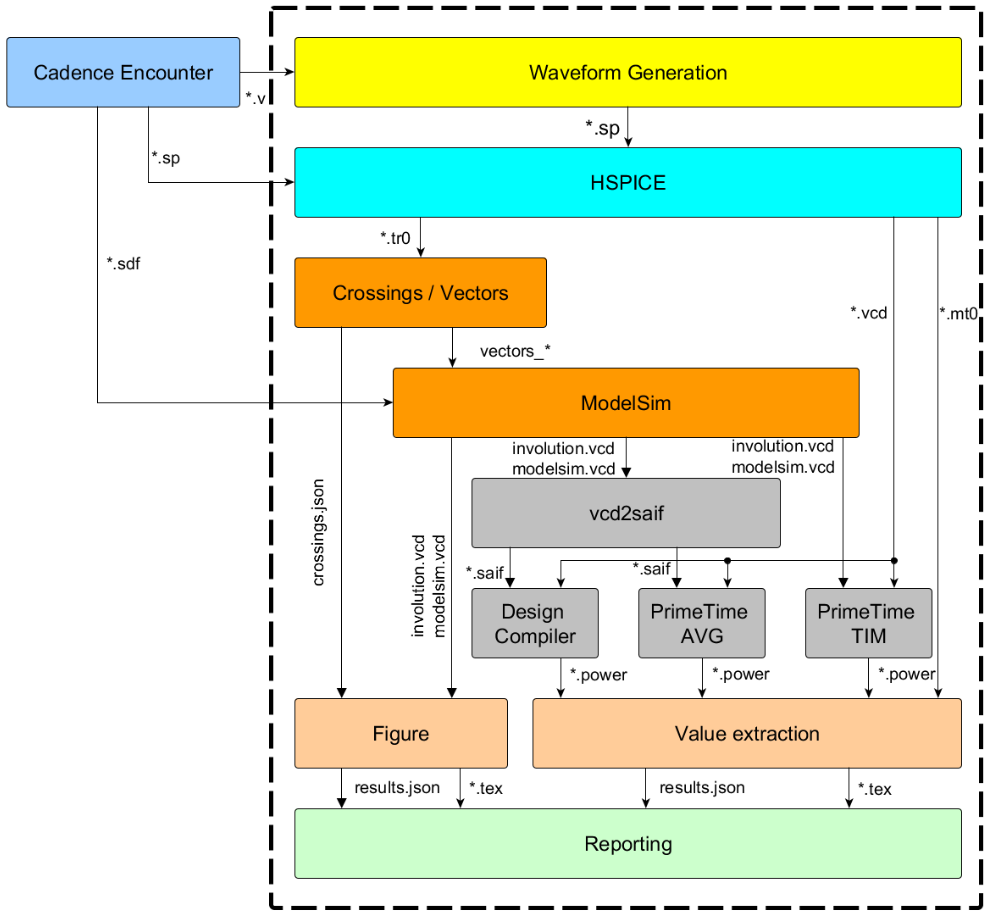

Figure 3.1: The workflow of the Involution Tool.

3.1 Waveform Generation

The waveform generation is the first step of the Involution Tool. For each input port of the circuit under test, a random test signal is generated. How the input signals are generated can be specified in a configuration file (see Section A.2.1). In a first step, the transition time of the next transition and the input port on which the transition should happen are randomly chosen. The time of the next transition is determined by a normal distribution, where the mean and the standard deviation can be defined in a configuration file.

The waveform generation also allows to group input ports together. If a transition is created for one of these input ports, we randomly decide for each other input port of the group if a transition should be generated, and if so, at which time. Again, the groups, correlation possibility, μ and σ can be specified in the configuration file. Grouping signals together can be useful, if we want to test multi-input gates. By grouping all the inputs of a gate together, we can increase the probability that there are transitions on each of the input ports in a narrow scope. Unfortunately, there is no specific implementation for multi-input involution gates yet.

This option not only allows to produce correlations between transitions on input ports, but also between an input port and an output port. Figure 3.2 shows an example circuit where this configuration option could be useful. The goal is to generate correlated transitions on the inputs of the NAND gate. We know that it takes some time until a transition at input port A is propagated to the output and finally reaches the input port of the NAND gate. Therefore we can specify a higher μ for the group in which the signals A and B are. Since we only want transitions at port A possibly cause correlated transitions at port B we can specify a one-way option for the group as well, which means that only the first signal in a group can cause correlated transitions at the other signals of the group.

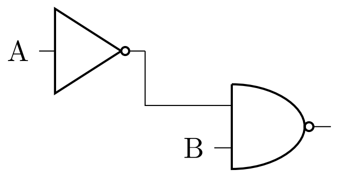

Figure 3.2: Example circuit where correlating transitions could be useful.

Another option for the waveform generation is how the time of the next transition is calculated. If the option *GLOBAL* is specified, the randomly generated time is added to the last transition at any input port in the circuit. If *LOCAL* is defined, only the time of the last transition on the input port where the new transition is added is considered. The latter option leads to denser transitions on the input signals.

A pseudo code for the waveform generation algorithm can be seen in Algorithm 3.1.

Algorithm 3.1: Waveform generation algorithm.

```
Input: configuration
curr_trans_count ← 0;
while curr_trans_count < trans_count do
    next_sig ← choose one signal randomly;
    time ← random($\mu$, $\sigma$);
    if next_transition_mode = GLOBAL then
        transition_time ← last_transition_time + time;
    else if next_transition_mode = LOCAL then
        transition_time ← last_local_transition_time + time;
    end
    transitions[next_sig] += transition_time;
    curr_trans_count++;
    foreach Group g in which the signal next_sig is contained do
        foreach Signal s in the group g except next_sig do
            if rand() < correlation_possibility then
                time ← random(group_$\mu$, group_$\sigma$);
                new_transition_time ← transition_time + time;
                transitions[s] += new_transition_time;
                curr_trans_count++;
                if curr_trans_count $\geq$ trans_count then
                    break;
                end
            end
        end
        if curr_trans_count $\geq$ trans_count then
            break;
        end
    end
end
Output: Input signal for all input ports
```

3.2 HSPICE - Simulation

The next step of the toolchain is required to make a fully fledged HSPICE [Syn13a] simulation of the circuit. The results of this simulation are used in later steps as reference. In the waveform generation step, the simulation template, which is a SPICE file that contains placeholders, of the circuit is populated with the generated input traces, and some other variables like V_{dd} and V_{th}. V_{dd} is the supply voltage of the circuit and V_{th} is the threshold voltage, which is used to differentiate between high and low voltage level. This template also contains the measurement settings. Currently, the average and maximum power consumption are measured per default, but the template can easily be extended to measure other significant values. Listing 3.1 shows how the power consumption can be calculated. First, the average current over the whole simulation time is measured and afterwards multiplied with the supply voltage. Another approach is to integrate over the current, multiply with the supply voltage and divide by the simulation time. Since integrating and dividing by the simulation time is probably exactly how the average calculation is performed internally, both methods lead to the same result. For the maximum power the absolute maximum current is measured and again multiplied with the supply voltage.

```
* Average Power calculation via average current
.MEAS TRAN avg_cur avg i(vdd) from=0ns to=<STOPTIME>NS
.MEAS TRAN pwr_avg PARAM='abs(avg_cur*V(mvdd))'
.print par('pwr_avg')

* Average power calculation method 2 via integral of the
↪  current
.MEAS TRAN total_cur integ i(vdd) from=0ns to=<STOPTIME>NS
.MEAS TRAN total_pwr_integ
↪  PARAM='abs(total_cur*V(mvdd)/<STOPTIME>e-09)'
.print par('total_pwr_integ')

* Maximum Power calculation via maximum current
.MEAS TRAN max_cur max 'abs(i(vdd))' from=0ns
↪  to=<STOPTIME>NS
.MEAS TRAN pwr_max PARAM='abs(max_cur*V(mvdd))'
.print par('pwr_max')
```

Listing 3.1: Example showing power calculation with HSPICE.

One configuration option for the circuit is to add inverter chains in front of each input port, in order to shape the input signals. Without them, Heavisides would be directly applied to the input ports, which does not model the reality very well. The user can choose how many of these sub-circuits should be used for each input, but usually two inverters in front of each input port proved to be sufficient. They use a different power supply, so that they do not distort the power consumption measurements of HSPICE. Note that adding inverter chains possibly leads to a decrease in the number of applied transitions at the actual input ports.

Not only power measurements are done with the HSPICE simulation, we also export the trace of each signal in the circuit (input, intermediate, output) in form of a value change dump (VCD), which is later used to simulate the power consumption with different tools than HSPICE (more information in Section 3.4). A complete example configuration can be found in Listing A.6.

3.3 ModelSim - Simulation

ModelSim [Men16a] is a digital simulation tool which is broadly used. The Involution Tool uses the program to simulate the HDL design of the circuit under test. Before we can start with the ModelSim simulation, we need to prepare the input stimuli for our circuit. Therefore we extract the result from HSPICE (*.tr0 file) and convert it into a condensed format. HSPICE saves the trace in a tabular-like manner where for time instants the corresponding analog value of the measured values is saved. Of course, this format is quite memory-intensive. Since we are only interested in the digitized version of the traces we can convert them. For each signal in the circuit we generate a list of times where a transition happens. The initial values are also saved in this intermediate format.

In a further step we transform the data from the intermediate format (crossings.json) into VHDL readable form (vectors_*). Of course we could have transformed the HSPICE trace directly into a VHDL readable version, but since we need the trace information at various other steps of the toolchain we decided to use an intermediate format. Furthermore, for the ModelSim simulation we are only interested in the traces of the input signals. Each input trace is stored in a separate file, which is read by the testbench during simulation. Since each of the input signals needs its own process we created a template for the testbench where the file read processes are automatically inserted before the simulation. Otherwise, the user would have to create a lot of copy-paste VHDL code.

We run the ModelSim simulation with two different delay models: The one implemented by ModelSim (further denoted as MODELSIM) and the involution model (INVOLUTION). Both of them rely on a standard delay format (SDF) file, which is generated by Cadence Encounter. This file specifies for each gate the delay for rising and falling edges. Also the interconnect between the gates is specified in this file. The output of the ModelSim simulation is a VCD file, which contains a list of transitions for each signal.

One thing we were not able to solve is how to apply the INTERCONNECT delay in the SDF file from the output of the gate to the output of the circuit. Usually we get a Verilog file from Cadence Encounter, containing the circuit, but when using this Verilog file we get the following warning message: "Invalid ports for INTERCONNECT." This is due to the fact that the second parameter of an INTERCONNECT has to be an input or a bidirectional port (see [Ovi95, pp. 48-49]), which is not the case in these Verilog files. Fortunately, all INTERCONNECTs in the circuits we used have been set to zero, and therefore this warning should not influence our simulation results.

We found two workarounds to the problem, but both are not really satisfying:

- Manually converting the Verilog file containing the circuit into a VHDL file. This is a tedious and error prone solution, but we decided to stick with it, until we find a real solution to the problem. One possible explanation why the SDF file annotation is working in combination with the VHDL file is that we have to manually create generics, whereas the annotation for Verilog files automatically matches SDF constructs to Verilog constructs (see Chapter 13 [Men16b]).
- Changing the type of each output from *output* to *inout* in the Verilog circuit file. This allows us to keep the Verilog file, and only make a slight modification. Since we are not sure which other implications these changes have on our simulation, we decided to not use this workaround.

We also experimented with the *sdf_annotate* task for annotating the Verilog circuit file. Unfortunately, this approach also led to the same warning message. Information on how this back-annotation method works can be found in [Men16b, pp. 470-471].

3.3.1 Involution gates

The involution channel implementation is the foundation for involution gates and is shown in Appendix B. Such channel is parametrized with the delay times for rising (tr_{01}) and falling (tr_{10}) edge, which can be found in the SDF file, and the pure delay (T_P). The delay times are used to calculate τ_{up} and τ_{do}, which are given by the following formulas:

$$\tau_{up} = \frac{tr_{10} - T_P}{-\ln\left(1 - \frac{V_{th}}{V_{dd}}\right)} \quad \text{and} \quad \tau_{do} = \frac{tr_{01} - T_P}{-\ln\left(\frac{V_{th}}{V_{dd}}\right)}$$

The actual calculation of the delay happens on a rising or falling edge on the input with the following two formulas (which are derived from the ones in Section 2.3):

$$\delta_{\uparrow}(T) = tr_{01} + \tau_{up} \ln\left(1 - e^{-\frac{T + tr_{10}}{\tau_{do}}}\right)$$
$$\delta_{\downarrow}(T) = tr_{10} + \tau_{do} \ln\left(1 - e^{-\frac{T + tr_{01}}{\tau_{up}}}\right)$$

After calculating the delay for the input transition, the output is set accordingly. Fortunately ModelSim already supports canceling of transitions. How the cancellation works can be seen in Figure 3.3. For the second pulse on the input, the calculated delay for the rising transition is larger than the the delay for the falling transition (which is in this case even negative). Therefore the transitions cancel each other. This is already supported out of the box by ModelSim, so the implementation does not need to keep track of previous transitions.

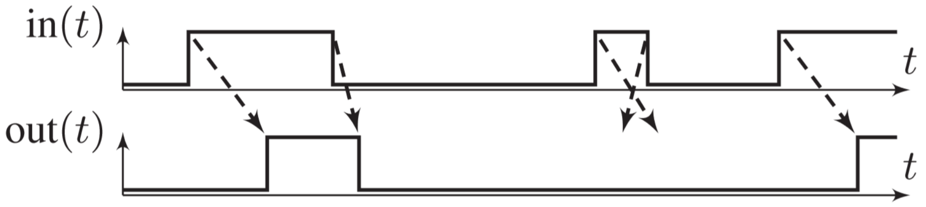

Figure 3.3: Figure showing the cancelation of pulses [FMN18].

Besides the two parameters for the rise (tr_{01}) and fall tr_{10} time of transitions, which are specified in the SDF file, there is a parameter for the pure delay T_P that can be chosen by the user. In Section 4.1.3, the influence of this parameter is discussed.

The gates use the involution channels as a building block. The Involution Tool allows the user to decide if the channel(s) should be placed at the input(s) or the output of the gate. There is also a third option possible, where the channels are placed at the output, but the delay times for rising and falling edge are swapped. This option is useful when simulating inverters (more information in Section 4.1.2). Note that involution channels and logic functions are always alternating, therefore all gates must use the same channel location. All gates also have to cope with the VITAL interface, which is necessary to pass the information from the SDF file to the gates. This means that the gates need to offer certain generics for each input and output (see [Men16b, p. 467]).

Basic gates, which consist of a simple logic function, can be automatically generated by the tool. The user can specify which gates should be generated, where the involution channel(s) should be placed, and the pure delay. This gate generation tool comes in handy when multiple simulation with different gate parameters shall be performed, as described in Section 3.7. However, since the gate generation is only able to create gates with simple logic functions, users still have to write more complex gates like AND-OR-Invert (AOI) and OR-AND-Invert (OAI) themselves. If manually written gates shall be used in combination with the multi execution tool, they must offer generics for the channel location and the pure delay, so that these parameters can be set from outside. The manual gate implementations have to place the involution channels according to the parameter and pass the specified pure delay on to the involution channels. These parameters can be passed at the start of the simulation. Note that the path to the input vector, which is required in the testbench, is already passed like this. Therefore it should be easy to extend this call.

3.4 Power Estimation

For estimating the power of a circuit, the following tools / options are supported:

- HSPICE
- Design Compiler [Syn10]
- PrimeTime [Syn13b, Loc16]
  - average based
  - time based

The power estimation made with HSPICE is used as a reference value for the other tools. The Design Compiler and the average based option of PrimeTime rely a so-called switching activity information file (SAIF) which is generated out of a VCD file and is a condensed version of the latter. These files only contain the time in the different states 0 (T0), 1 (T1) and X (TX) and the transition count (TC). The longer the signal stays in an unknown state (TX), the bigger is the negative impact on the accuracy of the power estimation. Information like the duration of pulses is completely disregarded. On the other hand, the time based simulation option in PrimeTime uses the VCD file, and has therefore more knowledge than the SAIF-based tools / options.

The power is estimated for the following three simulations:

- ModelSim standard delay model (MODELSIM)
- Involution delay model (INVOLUTION)
- HSPICE trace

The first two power estimations rely on the simulations that are generated during the ModelSim simulation, as described in Section 3.3. The third simulation relies on the trace generated during the HSPICE simulation, as described in Section 3.2, and is used to compare the power consumption result of HSPICE with the one we get with the above mentioned power estimation tools. In Chapter 4 we will see that there is a slight difference between these results.

3.5 Evaluation

Before the results can be evaluated, the results need to be extracted from the various output formats (Section 3.5.1). From these results, the power deviation between the analog simulation with HSPICE and the simulations with ModelSim is calculated (Section 3.5.2), and the traces of the different tools are compared (Section 3.5.3).

3.5.1 Value extraction

Due to the simulations with different tools, a variety of formats for the output are used. Our main goal is to extract as much information as possible from these output files, and bring the results into a standardized format. We decided to save the results in a json serialized dictionary, where the key represents the measured variable and the value represents the measured value. This format is suitable to store most of the results we achieve during simulation. We will discuss the exemptions in the following subsections.

Parsing the measure file (*.mt0) of HSPICE is quite easy, since the format is very regular. At first, we parse the names of the measured values and save them into a list. The next step is to save all the values into a list. Finally, we zip the names and the values together into a dictionary and convert the power measurement results to the same unit. This step is necessary, because in the report results from different simulation steps and different simulation tools are mixed, and therefore the power unit is not necessarily the same. HSPICE for example outputs the results in Watt, whereas Design Compiler shows the power in milli-Watt.

Unfortunately, parsing the output file of the power estimation tools is not easy. These files have a log file like structure, which makes it difficult to decide if the line is a log information or a result. These output files are divided into different sections, all of them starting with a certain structure. The following sections are parsed: general information, design, unit information, power report table, power total and peak power total. The second last section is only present in PrimeTime result files, and the last section is only present in PrimeTime time based simulation. Since this parsing method relies heavily on the structure of the output file, changes of the output file format in different releases of the power estimation tools can have major implications on the parser.

Moreover, we also store the environment variables in our unified data structure, because a lot of configuration of the Involution Tool is done via environment variables, and therefore we want to be able to access this information in the reporting. The waveform generation settings (see Section 3.1) are also stored in this unified data structure.

3.5.2 Power deviation calculation

One major benefit of the tool is that the power consumption of different models and different tools can be easily compared. In the following we use two different kinds of reference values and compare the results of the power estimation tools to two types of reference values:

- SPICE/HSPICE: This is the value which is measured by HSPICE during the simulation.
- SPICE/column: During the HSPICE simulation, the trace is exported and saved into a VCD file. This file is then further applied to the power estimation tools, and the result is used as reference value. SPICE/column indicates that the power

estimation is based on the SPICE trace and performed with the tool in the column name.

Note that the first type is just one value, whereas the second type produces three reference values, since each power estimation tool / option calculates one value. By having these two types, the actual deviation and the deviation caused because of different tools can be better distinguished. An example for the results of the power deviation calculation can be seen in Table 4.2. In the first row, both types are compared, and one can see a slight difference, which justifies the decision to use two different types of reference values. Based on these two reference values, the deviations for MODELSIM and INVOLUTION are calculated (row 2-5 in Table 4.2). The results are again stored in the dictionary containing all our simulation results.

3.5.3 Waveform comparison

In this step, the traces of HSPICE and ModelSim are compared with regard to the following metrics:

- Number of transitions
- Area under the deviation trace
- Number of suppressed and induced glitches

At first, the HSPICE traces are loaded from the previously generated intermediate format (see Section 3.3), and the traces from the ModelSim simulation are read from the VCD files. Next, the deviation for corresponding HSPICE and ModelSim waveforms is generated. While iterating over the transitions of both traces it is checked if the next transitions on both traces are at the same time instant and have the same direction. If two matching transitions are found, the next two transitions are compared. Otherwise a pulse is inserted in the deviation trace. Requiring that the transitions happen at the exact same time instant sometimes led to a deviation on the input trace, which should of course not be the case. This was due to the fact that the smallest possible time step in ModelSim is $1fs$, whereas HSPICE allows higher accuracy. Therefore, transitions which happen within a time range of $1fs$ are now considered equal. If the deviation trace between two signals is always 0, they match perfectly. This must be the case for input signals.

After generating the deviation trace, the result is optionally saved in a comma-separated value (CSV) file with some additional information. On top of the information about the length of the deviation pulses, it is also stored if the deviation pulse is due to a glitch on one of the two compared traces. This is important because not only the area under the deviation trace and the number of transitions are interesting for comparing traces, but also the number of induced and suppressed glitches is relevant (see below for the definition). Glitches are detected by checking if there is a complete pulse (two subsequent opposite transitions) on one trace while on the other trace the values stays constant during the pulse.

Figure 3.4 shows an example where the traces A and B shall be compared. For the first deviation ①, no glitch is detected, because there are no two subsequent transitions on one of the traces while the other trace stays constant. For ②, a glitch on Trace A is detected. Unfortunately, the glitch detection can create false-positives, which is showed in ③. At first, there is a short pulse on Trace A, then on Trace B. These two pulses do not overlap, and therefore the glitch detection decides that there is a glitch on Trace A, immediately followed by a glitch on Trace B. One possible solution for this problem would be to check the results afterwards, and if a glitch on Trace A is nearly immediately followed by a glitch on Trace B (or vice versa) we ignore these two glitches. But this could again lead to incorrect results, because the result showed in Figure 3.4 could be correct if the two pulses on Trace A and Trace B are triggered by different transitions on the input. As one can see, this is a very challenging task which might be improved in future work.

Moreover, assume that A is the reference trace by HSPICE, and B is by ModelSim: In the following Sections, a constellation like ② will be called *glitch suppression*, whereas ④ will be called *glitch induction*.

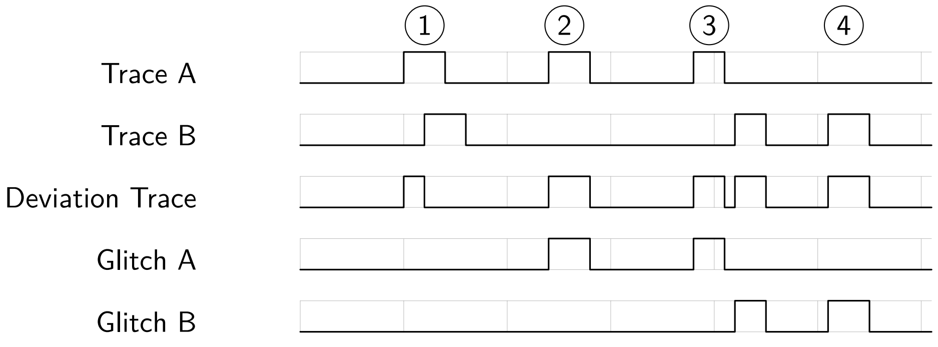

Figure 3.4: Comparison of different delay models.

3.6 Reporting

The reporting of the Involution Tool attempts to be very flexible and consists of several parts which can be combined and extended by the user. How these parts work together can be configured in a shell script. The default version consists of the following steps:

- Value extraction (Section 3.5.1)
- Power deviation calculation (Section 3.5.2)
- Waveform comparison (Section 3.5.3)

- Plotting (Section 3.6.1)
- Fill template and build report together (Section 3.6.2)

3.6.1 Plotting

During the waveform comparison, the following plots are generated:

- Trace of HSPICE, MODELSIM and INVOLUTION
- Deviation trace HSPICE versus MODELSIM
- Deviation trace HSPICE versus INVOLUTION
- Deviation trace MODELSIM versus INVOLUTION

For a higher number of transitions, the simulation duration can get quite long, and therefore the Involution Tool allows the user to create a definable number of zoom plots by specifying the number and how much they should overlap.

While iterating over each signal, the information for the report is prepared. The user can specify with a template how the information for one signal should look like, and the template is populated for each signal. After finishing the iteration, the populated templates for each signal are concatenated and saved. We decided to use templates for each row (signal), instead of adding the results to the dictionary, because it is easier to use the results in the report. Furthermore, various aggregated values, for example the total number of the glitches, the total area under the deviation trace, and the maximum transition count deviation (relative and absolute) are calculated during the iteration and saved in the results dictionary.

3.6.2 LaTeX template

The last step is to assemble all parts of the reporting together. A detailed description of the process can be found in the manual in Section A.2.3. In the top level template file, the sections which should appear in the report can be defined. Following sections are per default available:

- Basic information: Contains information about the environment, used libraries, simulation settings, information about the waveform generation, and also information about the gate configuration.
- Power consumption: Compares the power estimation results of the various used tools. Moreover, information about the peak power is listed.

- Waveform comparison: Lists the number of transitions for each simulation setting (HSPICE, MODELSIM, INVOLUTION) and for each signal. Also shows the area under the deviation trace as an indicator how different the two compared traces are, and information about the number of suppressed and induced glitches.
- Plots: The user can define plots which should be automatically added to the report file.
- Schematic: Adds a schematic of the simulated circuit.

All values extracted and saved during the previous steps are converted into a LaTeX readable version and can be used as variables throughout the report. Furthermore, all data with various length, for example information about the waveform for each signal and information about the gate configuration is filled into various sub templates which are included in the top level template file. Of course we could also save the information about the waveform and the gate generation in the result dictionary, but this would cause problems using the data in the report, since we cannot just access it with a single variable. Therefore we decided to use a template for a single entry (for example one signal, one gate, or one group for the waveform generation) and populate the template with one data set. Finally, the templates for all entries are put together and referenced in the report.

3.7 Multi execution

Since all previous steps are based on the random waveform generation, one pass through all simulations is of course not enough to make reliable statements about the overall performance of the involution delay model. Hence, we decided to add a tool to run the simulations multiple times and generate an overall report. The so-called *multi_exec* tool prepares everything for one simulation run and invokes the toolchain. This procedure is repeated until all configurations are processed. In the following section we call a complete execution of the toolchain a *simulation run*. Note that a simulation run consists of multiple simulations performed with HSPICE and ModelSim.

3.7.1 Configuration

The tool not only supports executing a simulation run multiple times with different random waveforms, it also supports sweeping over the following parameters:

(a) exp_channel_location: Is used to specify the location where the involution channel should be placed (for options, see Section 3.3.1).

(b) T_P: Concerns the gate generation algorithm and is used to parametrize the pure delay (T_P) of the involution channel. T_P and the exp_channel_location parameter currently only work, if the automatic gate generation is used. Ideas how to extend the Involution Tool in order to make manual implemented gates understand the parametrization can also be found in Section 3.3.1.

(c) waveform_generation: Is used to configure the waveform generation. The multi_exec tool takes the default configuration file and overwrites all settings which are specified in the multi_exec configuration file. This parameter enables the user to test certain group combinations for the input signals, but also simpler things like sweeping the parameters for the random function that generates the time to the next transition.

Sweeping over the previously described parameters in combination with multiple executions of a certain configuration enables the user to find good configurations for certain circuits and learn more about the involution channels. Information on how to configure these parameters can be found in the manual in Section A.4.1.

By default, the tool tries to keep the waveform between two subsequent simulation runs. This is possible for the first two parameters ((a), (b)), because they only concern the gate generation, that only has influence on the ModelSim simulation with the involution delay model and the steps afterwards. Keeping the waveform improves the performance, because we can skip the HSPICE simulation and also the ModelSim simulations of the HSPICE trace and the standard delay model. But the main reason why we want to keep the waveform between simulation runs is that it increases comparability. In order to maximize the number of simulation runs between which we can keep the waveform, one has to sweep over the parameters in a certain order: If one imagines sweeping over each parameter as a for-loop, it is obvious that those parameter for which the waveform can be kept have to be in the inner loops and the other parameters in the outer loops.

The easy approach for sweeping through the parameters can be seen in Algorithm 3.2. Unfortunately this algorithm is not easily extendable, since each new parameter that is added leads to a new nested for loop. One can image that this leads to a very nested structure for more parameters.

Algorithm 3.2: Multi execution simulation algorithm (simple approach).

```
foreach waveform in WAVEFORMS do
    generate_new_waveform(waveform);
    foreach channel in CHANNELS do
        foreach tp in TPS do
            simulate(waveform, channel, tp);
        end
    end
end
```

Therefore the alternative approach showed in Algorithm 3.3 has been implemented. It is based on a dictionary containing the parameters as key (to be more precise: integers that reflect the parameters) and as value the index to the current setting for each parameter.

After each simulation, the parameter dictionary needs to be updated. The parameter with the lowest key (in this case *TP*) is incremented. If all settings for this parameter have been simulated (check for *LENGTH[key]*), it is reset and the algorithm continuous to check the next parameter. When a parameter overflows, it is checked if the waveform can be kept. If for example *TP* overflows, the waveform can be kept, whereas this is not the case for *CH*. If the last parameter overflows, all possible combinations have been simulated and the simulation can be stopped, or a new iteration over all combinations can be started. The advantage of this approach is that new parameters only have to be added to the parameter and length dictionary. Depending on the type of parameter that has been inserted and its key, it is possibly necessary that the *last_key_to_keep_waveform* needs to be updated. Both pseudo-codes assume that the waveform should be kept as long as possible.

3.7.2 Reporting

When all simulations have been completed, the tool aggregates the results of each simulation run together. The user can specify properties which should be aggregated, and the tool calculates the minimum, maximum and average value for these properties. By default, results concerning the power estimation and the waveform generation are specified. It is also possible to use regular expressions for specifying the properties.

Similar to the reporting described in Section 3.6, the extracted and aggregated results can be used in the tex files as variables. The first step is to generate the LaTeX file which contains information about the configurations that have been used. By default, the report shows the average, minimum and maximum deviation for the power consumption in a tabular format. For the latter two, each cell not only displays the value, but also a link to the configuration with which this result has been achieved. Therefore the user can gain a quick overview which configurations achieve good results. The report also shows an aggregated version of the waveform comparison. It is also possible to list the results for certain properties for each simulation run in a sorted way. This rankings can be used to find out which configurations achieve the best results, concerning a certain property.

Since the amount of data that is generated during the simulation runs is quite large, the tool also supports a CSV export. It can be configured which parameter should be exported and in which order. This allows the user to group certain results together, for example results concerning the power consumption. The export makes it possible to process the extracted data with custom tools.

Algorithm 3.3: Multi execution simulation algorithm (dictionary based approach).

```
TP ← 1;
CHANNEL ← 2;
WAVEFORM ← 3;
param_dict[TP] = 0;
param_dict[CH] = 0;
param_dict[WV] = 0;
length[TP] = LENGTH(TPS);
length[CH] = LENGTH(CHANNELS);
length[WV] = LENGTH(WAVEFORMS);
keep_waveform ← false;
last_key_to_keep_waveform = CH;
while true do
    if not keep_waveform then
        generate_new_waveform(WAVEFORMS[param_dict[WV]]);
    end
    simulate(WAVEFORMS[param_dict[WV]], CHANNELS[param_dict[CH]],
      TPS[param_dict[TP]]);
    foreach key in sorted(param_dict) do
        overflow ← false;
        param_dict[key]++;
        if param_dict[key] == length[key] then
            overflow ← true;
            param_dict[key] ← 0;
            if key < last_key_to_keep_waveform then
                keep_waveform ← true;
            else
                keep_waveform ← false;
            end
        else
            break;
        end
    end
    if overflow then
        break;
    end
end
```

CHAPTER 4

Results

For the simulations reported in this thesis, we used two different test circuits: One based on inverters called *inv_tree* and another one based on 2-input NAND gates called *c17_slack*. The first circuit is a simple inverter tree, the second circuit is more complex and can be used to test how involution channels behave in combination with multi-input gates, and how different group combinations in the waveform generation affect the results.

The first simulation we did with each circuit was a simple comparison to the standard delay model. Further simulations focused on the effect that different parameters have on the involution delay model. In the following sections, the standard delay model still will be denoted as *MODELSIM*, and the involution delay model will be denoted as *INVOLUTION*. All simulations have been made with the multi_exec tool as described in Section 3.7 and if not otherwise stated the number of generated transitions $N_{trans} = 500$ and the number of executions of each configuration $N = 10$.

4.1 inv_tree

The first circuit we used for testing is an inverter tree. The schematic for the circuit can be seen in Figure 4.1. It has a single input *din* and four outputs *dout1* - *dout4*.

4.1.1 MODELSIM versus INVOLUTION

The first simulation was a simulation to compare the standard delay model versus the involution delay model. We fixed the parameters for the involution gates to $T_P = 1ps$ and put the involution channel at the input of each gate (before the boolean function). In Section 4.1.3 it is justified why setting $T_P = 1ps$ is reasonable. Table 4.1 shows the average of the estimated power consumption.

We observed that the power estimation we got from HSPICE slightly differs from the power estimation we received for simulating the HSPICE trace with the Design Compiler

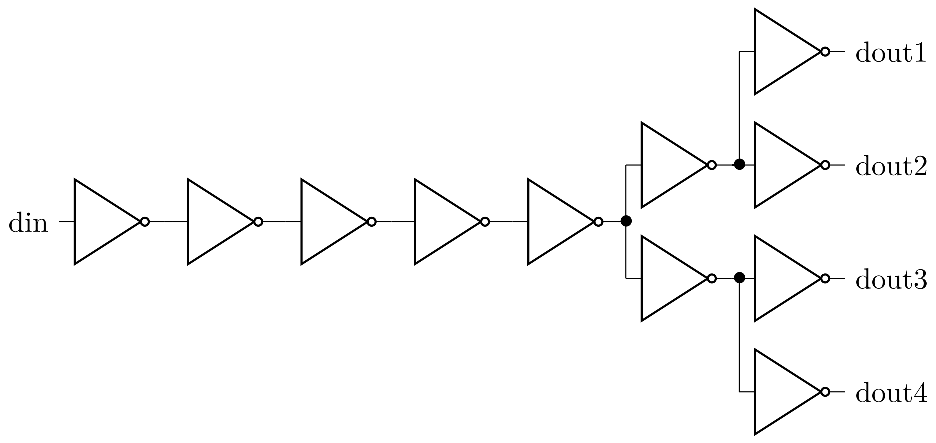

Figure 4.1: Schematic for the inv_tree circuit.

	HSPICE	DC	PT Avg	PT Time
SPICE	159.5	152.3	152.3	152.0
MODELSIM	-	244.8	244.8	244.7
INVOLUTION	-	186.9	186.9	179.5

Table 4.1: Average power consumption (in µW).

and PrimeTime. Therefore we decided to offer two reference values. The first reference value is the result we got from the HSPICE simulation (HSPICE) and the second one is derived by the corresponding power estimation tool (column). Table 4.2 shows the achieved results.

Reference	DC	PT Avg	PT Time
SPICE/HSPICE vs. SPICE/column	4.550	4.550	4.711
SPICE/HSPICE vs. MODELSIM	53.73	53.73	53.63
SPICE/column vs. MODELSIM	61.07	61.07	61.24
SPICE/HSPICE vs. INVOLUTION	17.21	17.21	12.59
SPICE/column vs. INVOLUTION	22.80	22.80	18.16

Table 4.2: Deviation of average power consumption (in %).

INVOLUTION is, for all simulation tools and options significantly better than MODELSIM. This can also be seen by looking at the number of glitches that are generated by both delay models. MODELSIM induced about 900 glitches per simulation for all signals that are measured. This means that we estimate far more transitions and therefore the estimated power consumption is larger. The number of glitches that are induced by INVOLUTION is about 300 per simulation, which is significantly less. One interesting result of the simulation with the involution delay model is that it also suppresses glitches from the HSPICE trace (see Figure 3.4, case ②). For this configuration, the number of suppressed glitches is about 20 per simulation run. This suppression of glitches leads to a reduction of the power estimation which in our case helps to increase the accuracy. This is still an unwanted effect, however, since we need to model the HSPICE trace as accurate as possible. Thus, only considering the power consumption may result in somewhat misleading conclusions. Only all metrics (power consumption, transition count, area under the deviation trace, glitches) together allow the user to draw conclusions about the simulation result.

4.1.2 Channel location

The next simulation was about comparing the influence of the location of the involution channels. We placed the channel before and after the boolean function and compared the results. We expected the exact same simulation results. If one considers the inverter gates as black boxes, the output trace should be the same for each gate, regardless of the location of the involution channel. Note that this assumption is only valid for single input gates like inverters or buffers. After the first simulation, the results were quite different, which was unexpected. The problem was that the rise and fall time, which are specified in the SDF file, are not symmetric, and therefore the channel location makes a difference. We implemented a third channel location called OUTPUT_SWAPPED, where the channel is placed at the output, but the rise and fall time are swapped (see Section 3.3.1). When comparing this new channel location with the channel at the input, we got the exact same results, as initially expected.

4.1.3 Pure delay

Next, we examined the influence of the pure delay parameter on the involution delay model. We simulated with three different pure delays $T_P = \{0.5ps, 1.0ps, 1.5ps\}$. Figure 4.2 shows the involution delay function $\delta(T)$ with fixed rise and fall times $t_{01} = t_{10} = 10ps$ and different pure delays T_P.

One thing we noticed immediately was the increase of the power consumption and the transition count with increasing pure delay. In this case, an increase also meant a higher deviation from the reference value. Since the values of these two metrics increased, we also expected the area under the deviation trace to increase, but this was interestingly not the case. With increasing pure delay, the number of induced glitches in the involution trace increased and the number of pulses from the HSPICE trace that have been suppressed in the involution trace decreased. These two effects both explain the increasing power consumption and the increasing transition count. One possible explanation why the number of glitches increases with increasing T_P is that the involution channels become more sensitive, in the sense that pulse cancellations are less frequent.

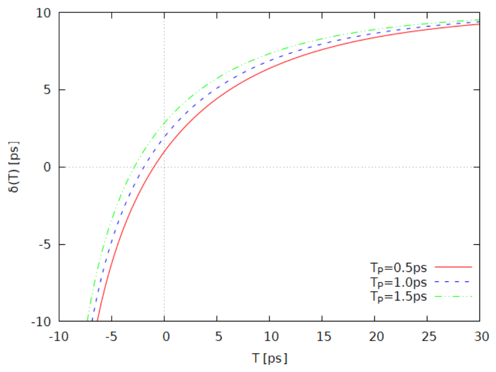

Figure 4.2: Involution delay functions with different T_P.

4.2 c17_slack

The second test circuit is based on 2-input NAND gates, and can be seen in Figure 4.3. Currently, the tool only supports single input involution channels, which disregard the subtle behavior of multiple inputs. The used NAND gate either consists of two involution channels at each input or of one involution channel at the output, depending on the configuration of the gate generation.

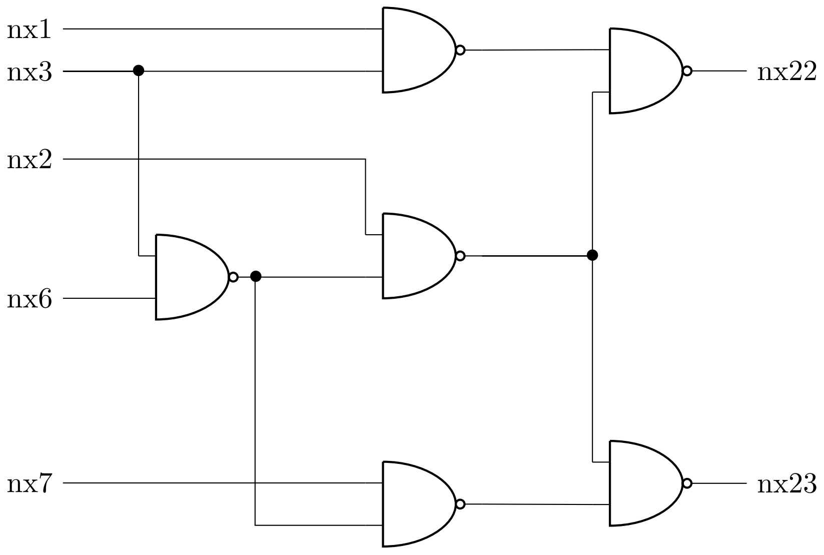

Figure 4.3: Schematic for the c17_slack circuit.

4.2.1 MODELSIM versus INVOLUTION

This simulation aims at comparing the standard delay model with the involution delay model. We used a pure delay $T_P = 1ps$ and placed the involution channel at each input of the NAND gates. Table 4.3 shows the average power consumption, and Table 4.4 shows the deviation of the power consumption from the reference value simulated with HSPICE respectively the HSPICE trace, processed via the tool accordingly to the column.

	HSPICE	DC	PT Avg	PT Time
SPICE	54.72	55.74	55.74	55.65
MODELSIM	-	63.03	63.03	62.99
INVOLUTION	-	65.54	65.54	62.28

Table 4.3: Average power consumption (in µW).

Reference	DC	PT Avg	PT Time
SPICE/HSPICE vs. SPICE/column	2.013	2.013	1.806
SPICE/HSPICE vs. MODELSIM	15.14	15.14	15.07
SPICE/column vs. MODELSIM	13.16	13.16	13.23
SPICE/HSPICE vs. INVOLUTION	19.76	19.76	13.80
SPICE/column vs. INVOLUTION	17.70	17.70	11.98

Table 4.4: Deviation average power consumption (in %).

The standard delay model has a slightly better performance than the involution delay model. When comparing the deviation values of this simulation with the results from Section 4.1.1, one can see that the performance of the standard delay model significantly improved, while the performance of the involution delay model remains nearly constant. One explanation for this behavior is that the number of glitches has reduced drastically. While we had about 900 glitches in the inv_tree simulation for the standard delay model, we now have about 40 glitches per simulation. The involution delay model generates about 50 glitches per simulation. One reason for the reduction of the number of glitches is that it is harder to generate glitches in NAND gates, since the inputs have a masking effect.

Figure 4.4 shows the switching waveforms for a NAND gate, simulated with HSPICE. The first two plots show the inputs A and B, the third plot shows the output Z, and the last plot shows the digitalized version of the output. The falling transition at $1ns$ on A causes a rising transition at the output. At approximately $2ns$, both inputs nearly simultaneously have a falling transition. This again causes a rising transition at the output. We see that this time the transition is significantly faster. Also when we look at the last plot, we can see that the threshold value is reached earlier in the second case. Such effects are currently not considered in our involution channel implementation, which is a possible explanation why the involution delay model underperforms for multi-input gates.

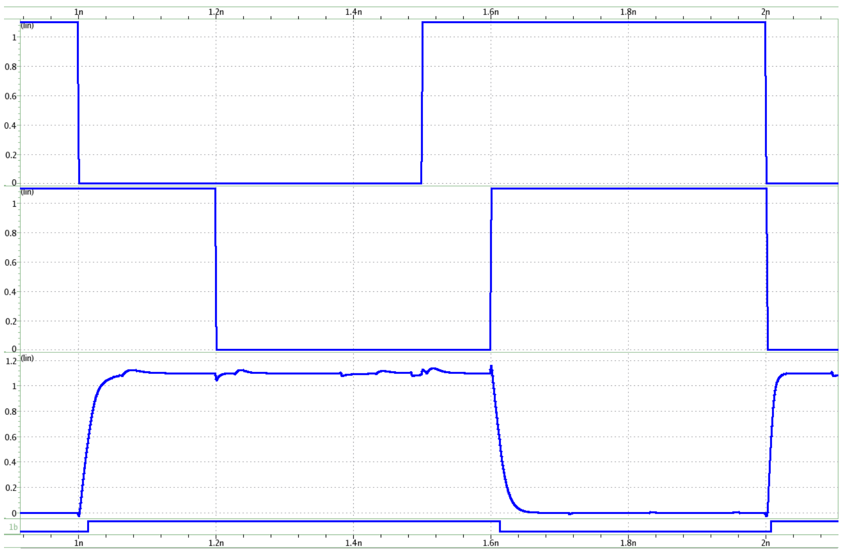

Figure 4.4: Switching waveforms for a NAND gate.

4.3 General remarks

4.3.1 Peak power

The time based power estimation option of PrimeTime also allows to estimate the peak power and the time of the peak power. Unfortunately the results of HSPICE and PrimeTime diverged significantly. For some simulation settings, even the power estimation made with the HSPICE trace overestimated the power by a factor of two. Since there was such a huge difference in the simulation results, we decided to not use the peak power as a metric for comparison.

4.3.2 Simulation performance

The tool also allows to keep track of the time that each simulation step takes. Table 4.5 shows an overview for the two circuits, each with 500 transitions at the input in total. It can be seen that the most time consuming task is the HSPICE simulation, which is the main reason, why we conduct research on delay models. Next is the reporting task, where especially the waveform comparison is rather time consuming. The third most time consuming task is power estimation, which is due to the fact that three different models, each with three different options, are simulated. Once the involution delay model is established, one would only use one model and one option. Therefore the corresponding simulation time would become negligible.

The overall time for a complete simulation run is about three minutes. If the waveform can be kept from the previous run (see Section 3.7), the simulation time decreases significantly to approximately one minute, since only tasks indicated with (*) need to be executed again.

circuit	inv_tree	c17_slack
# of transition	500	500
simulation duration	16.0	15.0
generate	0.1	0.1
hspice	67.9	84.2
crossings	17.3	22.5
read (vectors)	0.1	0.4
ModelSim	6.4	6.3
involution (*)	1.9	1.8
modelsim	4.4	4.5
power	34.7	35.1
modelsim	11.4	11.9
DC	4.9	5.1
PT AVG	3.2	3.5
PT TIM	3.3	3.3
involution (*)	11.6	11.6
DC	5.1	5.0
PT AVG	3.3	3.2
PT TIM	3.3	3.3
hspice	11.3	11.6
DC	3.4	3.2
PT AVG	4.8	5.2
PT TIM	3.1	3.2
report (*)	55.8	39.5
keep waveform / only involution	69.4	52.9
total	182.0	188.1

Table 4.5: Simulation performance (duration in s)

Since the time based power estimation option of PrimeTime uses significantly more information than the average based option of PrimeTime and the Design Compiler estimation, we thought that the simulation also might take longer. Interestingly, this was not the case, as also revealed in Table 4.5: Both simulation options of PrimeTime need approximately the same amount of time, and the Design Compiler seems to be slower.

CHAPTER 5

Conclusion and Future Work

The main result of this Bachelor's thesis is the Involution Tool, which allows to simulate the involution delay model and compare its results to SPICE simulations and the standard delay model of the simulation tool ModelSim. The focus was on supporting a variety of different configuration options for parametrizing the toolchain. Another highlight of the tool is the extraction of as much as possible information from the different output formats and integrating and presenting it in a clear, consistent and user-friendly format.

To demonstrate the capabilities of the Involution Tool, we performed simulations of two different designs, an inverter tree and a multi-input circuit. Since there is no theoretical model for multi-input involution channels yet, the involution model did not outperform the standard delay model significantly for the latter. For the inverter tree, however, achieved results showed a significant performance increase: The deviation of the power consumption estimation could be decreased from approximately 53% to 15%. Also the number of induced glitches is significantly lower when using the involution delay model.

A nice feature for the tool would be to support INTERCONNECT delays. As described in Section 3.3, we still have not been able to find a satisfying fix for a warning regarding interconnects. As soon as we want to model the delay time of interconnects between the output and input of another gate, we also have to adapt our involution gates, such that they incorporate these interconnect delays. Another shortcoming of the current implementation are limited possibilities to incorporate manually designed gates.

Possible future work includes the implementation of multi-input involution channels and their simulation. Another interesting extension would be different thresholds for rising and falling transitions, which need substantial adaptions of the Involution Tool as well. The currently used glitch detection could be also improved, since the implemented model is only a good approximation. Another nice feature would be to support multiple types of involution channels: Currently, only the exp-channel is implemented, but there are many conceivable alternatives.

APPENDIX A

Manual for the Involution Tool

A.1 General

A.1.1 Structure of the Involution Tool

The basic structure of the involution tool is split into the following parts:

```
│ circuits
│ ├── inv_tree
│ ├── c17_slack
│ ├── config.cfg
│ └── gate_config.json
├── experiments_setup
│ ├── python
│ ├── scripts
│ ├── spice
│ ├── tex
│ ├── vhdl
│ └── Makefile
├── manuals
└── tools
```

Circuits

This is the folder where new circuits should be added. *inv_tree* and *c17_slack* are two basic circuits which can be used as "template" for new circuits. The *config.cfg* file contains variables which are used during the simulation process, and are used per default by all circuits. These variables can be overridden in the Makefile of the circuit itself, more information can be found in Section A.3.1. Most of the variables in the *config.cfg* file are used for defining the folder structure of a specific circuit.

The recommended (and default) structure is:

- input: contains the generated waveform saved in a json file and main_new_exp.sp, a spice file where the placeholders (for example for the input traces) are already replaced.
- hspice: contains the trace and logfiles of the HSPICE simulation.
- crossings: contains a json file containing all the extracted crossings from the HSPICE trace (*.tr0) file.
- vectors: contains the prepared input trace for each input port. This trace is extracted from the crossings.json file and converted into a VHDL readable version.
- modelsim: contains the logfiles, and especially the vcd files, which are later used for power estimation and plotting.
- power: contains the logfiles from the power estimation tools and the generated reports. Also contains the scripts (with replaced placeholders) which are executed by the power estimation tools.
- results: contains the reports for each simulation in a subfolder. Such a subfolder contains all generated figures (showing and comparing the different traces and their deviation to the HSPICE reference trace), latex files and results extracted from the various result files. In case of using the multi_execution tool, the reports can also be found in this directory.
- gates: simple gates can be created automatically (more information in Section A.2.2). The resulting *.vhd files are saved in this folder.

The described folders contain the results of the different steps of the simulation process. More information on the files required for adding a new circuit can be found in Section A.3.1.

Experiment setup:

This folder contains a set of scripts and templates which are used during the default simulation process. Each element of the toolchain can be adapted to the needs of the user (by changing the Makefile of the specific circuit, and overriding the parts of the toolchain that should be handled differently for the circuit). The Makfile calls these scripts in several steps. More information on the Makefile and the workflow can be found A.1.2.

- python: contains python scripts used for various steps throughout the whole workflow (generating waveforms, parsing trace files, plotting figures, preparing latex reports, . . .).

- scripts: scripts and templates which are used for preparing the actual scripts used by ModelSim and the power estimation tools.
- tex: contains templates for the automatic report generation, more information in Section A.2.3.
- spice: contains useful “snippets” which can be added to the spice file of the circuit. Currently only contains *shaping.sp*, which can be used for adding an inverter chain at each input.
- vhdl: contains the involution delay channel implementation, templates for the testbench generation and a folder called *gates*, where more complicated gates, which can not be automatically generated, can be added. It also contains an implementation for a pure delay channel, which is currently not used.

Manuals

Most of the manuals in this folder are for HSPICE. Unfortunately there are no real manuals for Design Compiler and PrimeTime, so the user has to stick with the man pages.

Tools

This folder contains tools which can be used in combination with the “basic” Involution Tool. Currently only the multi_exec tool is available, more information in Section A.4.1.

A.1.2 Workflow of the Involution Tool

The flowchart in Figure A.1 shows the workflow of the Invoultion Tool.

The Involution Tool is split into several parts, each part builds upon the previous parts. The flowchart shows the main steps for a simulation. Each of the parts corresponds to one or more Makefile recipes.

Waveform Generation

Makefile recipes: *generate*

Responsible for generating a new input waveform, according to the configuration in the directory of the simulated circuit. More information on the configuration can be found in Section A.2.1. This section also prepares the *.sp file which is later used by HSPICE (replacing placeholders and inserting the generated waveform)

HSPICE Simulation

Makefile recipes: *hspice*

Uses the previously generated *.sp file and runs the simulation.

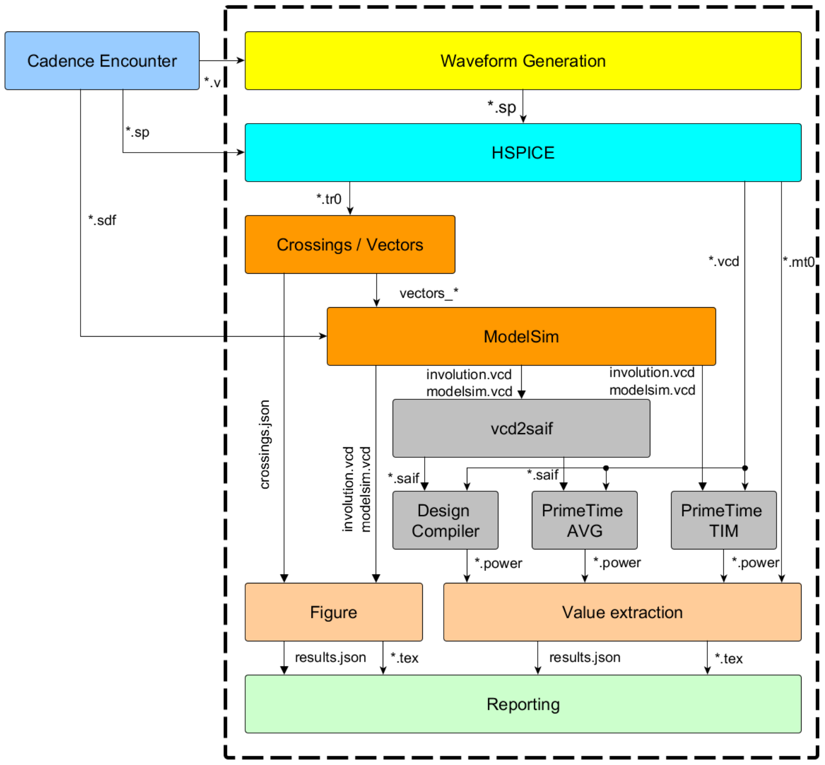

Figure A.1: The workflow of the Involution Tool.

ModelSim

Makefile recipes: *crossings, read, gates, sim*

This part parses the trace file of the HSPICE simulation (*.tr0), and generates the vector files, which are later applied through the testbench. Since simple gates can automatically be generated (see Section A.2.2), we also need to create these gates before the ModelSim simulation starts. The actual simulation uses *.do files which have been prepared in previous steps, and generates *.vcd files.

Power Estimation

Makefile recipes: *power (power_spice, power_dc, power_pt)*

The power estimation is currently done by two tools: Design Compiler (DC) and PrimeTime (PT). DC only supports an "average-based" mode, and needs a *.saif file. The switching activity file (*.saif) is generated from the value change dump (*.vcd) file from the ModelSim simulation. It reduces the information from the *.vcd file to a minimum. PT also supports another mode, the so called "time-based"-mode, which uses "more" information than the "average-based" mode, because it also considers when the transitions are occurring. Therefore, the “time-based” mode can also estimate the peak power, and not only the average power.

Reporting

Makefile recipes: *report*

Since the reporting process is highly customizeable, a shell script is called for generating the report. This script can be easily adapted by the user. By default, the script first extracts information from the various result files of the used tools. The goal is to save this information in a unified format. The resulting format is a dictionary that is json serialized and stored into a file called *results.json*.

Another step in the script is to generate different figures, displaying the traces of the different delay channel types (ModelSim, Involution) and the reference trace from HSPICE. This step also calculates the deviation between the HSPICE trace (crossings.json) and the different simulations and generates plots for the deviation. After generating the deviation traces, the tool also calculates the number of glitches and stores it together with further information about the deviation into a CSV file.

Sections where the output is configuration dependent (length of table, repeating structure with different data) are generated in a further step. In this step, template files are used so that the user can for example configure how one row of a table should look like. These template is used for each row and finally put together.

The final result of the reporting step is a folder containing all the required *.tex files, the extracted information (*results.json*), the generated plots and the CSV files containing the information about the deviation. The folder also contains the random waveform that has been used for simulation, so that the results can be reproduced if necessary.

A.2 Configuration

A.2.1 Waveform generation

The following snippet shows an example configuration for the waveform generation:

```
  "N": 500,
  "calc_next_transition_mode": "GLOBAL",
  "groups": [
    {
      "correlation_possibility" : 0.4,
      "mue": 0.0025,
      "oneway": false,
      "sigma": 0.0085,
      "signals": [
          "nx3",
          "nx6"
        ]
      },
      {
        "correlation_possibility" : 0.6,
        "mue": 0.01,
        "oneway": false,
        "sigma": 0.03,
        "signals": [
          "nx1",
          "nx3"
        ]
      }
    ],
    "mue": 0.029,
    "sigma": 0.01,
    "signals": [
      "nx1",
      "nx7",
      "nx3",
      "nx2",
      "nx6"
    ]
```

Listing A.1: Example configuration for waveform generation.

Parameters

- N: the overall number of input transitions to generate.
- calc_next_transition_mode:
  - LOCAL: The randomly generated transition time is added to the time of the last transition on the current signal.
  - GLOBAL: The randomly generated transition time is added to the time of the last transition on any input signal.
- mue / sigma: Used for parametrizing the normal distribution, which is responsible to generate the time of the next transition (delta time since the last transition).
- signal: A list of signals for which input waveforms should be generated.
- groups: Groups can be used for specifying correlations between two or more signals. Groups can be useful for example, if two input signals are applied at the inputs

of a gate. Since we want to generate short pulses, it can be useful to generate transitions at these input signals which are somehow correlated.

- mue / sigma: These two values are used for parametrizing the random number generation, which is used for calculating the transition time of the “following” transition. The random numbers are distributed according to a normal distribution.
- signals: Contains a list of two or more signals. If a transition happens on any signal in the list, each other signal in the list is checked if a “following” transition should be generated.
- correlation_possibility: This parameter specifies the possibility that a “causing” transition causes a “following” transition on any of the other signals. This value is used for randomly deciding for each signal if a “following” transition should be generated
- one_way: If only signal A should cause transitions on the signals B, C, ... but not vice versa, this option can be set to true. In general, this option can be useful if one input signal (A) is delayed (maybe because A is first inverted), and the other input signal (B) is not delayed, and these two signals are applied at a gate. Then it can be useful that only transitions on signal A cause transitions on signal B.

A.2.2 Gate generation

Basic gates can be automatically generated by the Involution Tool. The configuration file is placed in the general circuit directory and called *gate_config.json*. The basic configuration can be overriden by a file with the same name which is placed in the top level of the specific circuit.

Configuration

The following code snippet shows a basic configuration for a NAND gate with two inputs and one output:

```
"ND2M1N": {
   "T_P": 1,
   "exp_channel_location" : "INPUT",
   "entity_name": "ND2M1N",
   "function": "nand",
   "inputs": [
     "A",
     "B"
   ],
   "outputs": [
            "Z"
        ]
    }
```

Listing A.2: Configuration for a NAND gate with two inputs.

The configuration file is a dictionary, where the key is the name of gate, in our case *ND2M1N*. Several properties can be specified:

- T_P: Specifies the pure delay in ps, used for the exp_channel.
- exp_channel_location: The following options are available: “INPUT”, “OUTPUT”, “OUTPUT_SWAPPED”. Either the exp_channel is placed at the input (for each input one channel, before the combinatoric function) or at the output (for each output one channel). “OUTPUT_SWAPPED” swaps the tr01 and tr10 times from the sdf file. This setting is useful for inverters. If we use the same waveform and compare “INPUT” and “OUTPUT_SWAPPED”, we receive the same results. This is not always the case if we compare “INPUT” and “OUTPUT”, because of the possibly asymmetric rising and falling times of the gate.
- entity_name: Basically the same as the key.
- function: Currently only basic logic function are supported like: *and, or, nand, nor, xor, xnor, not.*
- inputs: Specifies the names of the inputs of the gate.
- outputs: Specifies the name of the output. Currently, only exactly one output is supported by the gate generation tool.

Non-configurable gates

Since the Involution tool can only configure very basic gates, more complex gates can be added in the folder experiment_setup/vhdl/gates/. These gates can then be used by all circuits. If there is a gate with the same name in the global gate folder and the circuit specific gate folder, the gate from the circuit specific folder is used.

Possible future improvements

- Generation of more complex gates like AOI
- Allow delay_input AND delay_output at the same time, and specify the used architecture in the simulation scripts
- Improve specification of outputs, so that multiple outputs are allowed (useful if we want an additional output which is inverted)

A.2.3 Report generation

The report generation of the Involution Tool attempts to be very flexible. The main parts of the report generation are:

- Scripts for extracting data from the reports of the various tools (HSPICE, Design Compiler, PrimeTime, . . .)
- Latex templates for displaying the data in a customer-defined way
- Scripts for generating latex Files, based on latex templates. Especially important for data that is depending on the configuration of the circuit and the circuit itself and has no “fixed” length and is recurring.
- Plots

Scripts for extracting data

With various scripts, the data of the simulation is extracted from the reports of the tools (HSPICE, Design Compiler, PrimeTime, . . .) and stored in a dictionary. Unfortunately, some of the tools do not print the data in a parser-friendly way. Therefore it is quite difficult to write stable, version independent parsers for the reports. It can happen, that the user has to adapt these scripts, for different versions of the tools, especially the output format for the Design Compiler and PrimeTime report are difficult to parse. For each file, a prefix is specified per default. This avoids conflicts in the dictionary containing all the parsed information. It also helps the user to find out how certain information from the tool report is named in the data dictionary from the Involution Tool. The extracted data is saved in the results folder of the specific circuit in the file *results.json.* This file is then converted to a file called *variable.tex*, which is included by the top level report file. All the specified variables can be used in the report files.

Latex templates

The Involution Tool supplies basic latex templates, which can be easily adapted to fit the users needs. The templates are located in the experiment_setup/tex/

- report_single.tex: top level file containing which includes all the generated tex files. Basic information which needs no additional processing can also be included in this file. There are some placeholders which can be used for specifying that a certain file should be included here.
  - %##VARIABLES##%
  - %##CWG##%
  - %##PLOT##%

  - %##WAVEFORM##%
  - %##SCHEMATIC##%

- cwg.tex / cwg_group.tex: These two files are used as template for displaying information about the waveform generation (see Section A.2.1). *cwg.tex* contains general information and can use variables from the data dictionary. The placeholder *%##GROUPS##%* can be used for specifying that the information about the group configuration should be inserted here. The configuration for one single group is in the file *cwg_group.tex*. There are some placeholders which can be used in this file:

  - %##SIGNALS##%: displays the signals which are in this group
  - %##SIGMA##%, %##MUE##%: The parameters used for randomly calculating the delay for a following transition that is caused by the initial transition on one of the signals of the group
  - %##ONEWAY##%: displays a plain bool value if the group is a one-way group or not
  - %##ONEWAYCHECKBOX##%: create a checked or unchecked checkbox which indicates if the group is a one-way group or not

- figure_group_template.tex / figure_template.tex: These two files specify how the defined figures in the config file should be displayed. *figure_group_template.tex* contains the layout for one “row” of figures. It can be useful to place multiple figures in one row, and this file specifies how this is done. The placeholder %##FIGURE##% indicates that the latex code for one figure is inserted here. The placeholder %##GROUPCAPTION##% can be used for adding captions to the row. If there are multiple rows, all rows except the last one get a *phantomcaption*. After the last row of figures the caption of all rows is inserted. We use phantomcaption here, because we only want one caption after all specified figures, and this package ensures that the numbering of the figures is correct.

- waveform.tex: Template for the table that shows the transition count of the different simulations for each signal. The placeholder %##LINES##% can be used for indicating that the rows of the table should be inserted here.

- schematic.tex: Specifies how the schematic of the circuit is displayed. The placeholder %##SCHEMATIC_PATH##% can be used to specify the path to the image of the schematic.

Report config file

The *report.cfg* file has to be placed in the top level of each circuit. The following code snippet shows an example configuration:

```
DYNAMIC_POWER_UNIT = 1e-6
LEAKAGE_POWER_UNIT = 1e-12
FIGURES = c17_slack_nx22.png; c17_slack_nx22_diff.png;
 ↪  c17_slack_nx23.png; c17_slack_nx23_diff.png;
SCHEMATIC_PATH = schematic.png
```

Listing A.3: Configuration for the reporting.

Since the different tools report the power consumption in different power units, it can be specified how the information should be displayed in the report. The power unit in which the data is displayed in the reports is parsed, and the extracted power information is converted to the specified power unit. With *FIGURES*, the figures which should be displayed can be specified. If a schematic for the circuit should be added to the report, the path to the schematic can be specified with *SCHEMATIC_PATH*. For more information on how to create the schematic of a circuit, see Section A.3.2.

Plots

During the report generation a number of plots is generated, based on the result traces of HSPICE and ModelSim. In the config.cfg (either in the general, or in the circuit specific config file) a number of parameters can be set:

```
### REPORT CONFIGURATION ###
export REPORT_CONFIG:=$(TOP_DIR)/report.cfg
export FIGURE_ZOOM_NUMBER=1  # < 2
export FIGURE_ZOOM_OVERLAPPING=0.1
```

Listing A.4: Configuration for a NAND gate with two inputs.

- REPORT_CONFIG: Path to the report.cfg file.
- FIGURE_ZOOM_NUMBER: Configures the number of zoom plots that should be generated. If the number specified is smaller than two, no zoom plots are generated.
- FIGURE_ZOOM_OVERLAPPING: Configures how much two adjacent zoom plots should overlap.

The following two environment variables specify, whether a csv file with information about the deviation trace should be generated during the figure generation process. These files can be useful when analyzing the deviation trace in detail. The csv files can be found in the same folder as the figures.

```
export FIGURE_INV_EXPORT_DEV_TRACE_INFO=True
export FIGURE_MSIM_EXPORT_DEV_TRACE_INFO=True
```

Listing A.5: Deviation trace export configuration.

A.3 How-To

A.3.1 Add a new circuit

For adding a new circuit, various configuration files have to be added. These files can be divided into the following two categories:

Circuit files

- circuit.vhd: Contains the basic structure for the testbench, required by ModelSim. It instantiates the unit under test, contains the signals and the most important thing is the placeholder "##INPUT_PROCESS##". During the simulation process, this placeholder is replaced by the process which applies the generated waveform to the input of the circuit. For each input, one such process is added to the file.

- *.sp: file containing the circuit under test, defined as a subcircuit, which is used in main_new.sp

- main_new.sp: This file is the "main"-file for the HSPICE-simulation. It includes the *.sp file containing the circuit under test.
  Following placeholders can be used:

  - <TEMP>
  - <VDD>
  - <VTH>
  - <STOPTIME>: This is the stoptime of your simulation. It is set depending on the generated waveform, shortly after all input transitions are over.
  - <signal_name>: This placeholder is replaced by the generated waveform for this specific input. For each input you should at least specify one placeholder, so that an input is applied to each input port.

  The file also contains all measurements, like the average power and the peak power. It is recommanded to name the parameters "pwr_avg" and "pwr_max", because the reporting utility looks for parameters with these names.

  Another option that can be activated is shaping. Since the input signals that are generated by the waveform generation are Heavisides, and therefore not realistic, one can add an inverter chain at the beginning of each input. These shaping inverters use a different supply voltage, because otherwise the power of the inverter chain would affect the overall power. How to use shaping and the perform power measurements can be seen in the following Listing.

```
* circuit: inv tree
.LIB <HSPICE_LIB>
.INCLUDE <HSPICE_CIR>

* main circuit
.INCLUDE ../inv_tree.sp

* Circuit for shaping the input
.INCLUDE ../../../experiment_setup/spice/shaping.sp

.TEMP <TEMP>
.OPTION
+ INGOLD=2
+ PARHIER=LOCAL
+ POST=CSDF
+ PROBE
+ BRIEF
+ ACCURATE
+ ABSVAR=0.05
+ DELMAX=100fs

* vdd
vdd mvdd 0 <VDD>v

* shaping
vddshape shapevdd 0 <VDD>v
Xmyshape1 dinshape din shapevdd shaping
.PROBE TRAN v(dinshape)

* circuit under test
Xmycir din dout1 dout2 dout3 dout4 mvdd inv_tree

* input
* use if no shaping should be at the input
*Vgpwl din 0 PWL(<din>)
* use if shaping should be at the input
Vgpwl dinshape 0 PWL(<din>)



.PROBE TRAN v(din) v(dout*) v(xmycir.g*:a) v(dinshape)
.TRAN 0.01PS <STOPTIME>NS
```

```

* Average Power calculation via average current
.MEAS TRAN avg_cur avg i(vdd) from=0ns to=<STOPTIME>NS
.MEAS TRAN pwr_avg PARAM='abs(avg_cur*V(mvdd))'
.print par('pwr_avg')

* Maximum Power calculation via maximum current
.MEAS TRAN max_cur max 'abs(i(vdd))' from=0ns
↪  to=<STOPTIME>NS
.MEAS TRAN pwr_max PARAM='abs(max_cur*V(mvdd))'
.print par('pwr_max')

* Find out the threshold values which are used by
↪  SPICE? -> we decided to use a single threshold, the
↪  same that we use for creating the crossings file
* temp1, temp2, temp3, temp4, temp5, temp51, temp52
.LPRINT(<VTH>, <VTH>) v(din) v(dout*) v(xmycir.g11:a)
↪  v(xmycir.g12:a) v(xmycir.g13:a) v(xmycir.g14:a)
↪  v(xmycir.g15:a) v(xmycir.g17:a) v(xmycir.g19:a)

.END
```

Listing A.6: Example configuration for SPICE containing measurements and shaping.

- *.v: Contains the verilog module for the circuit under test. Unfortunately, there is still a problem with INTERCONNECTs from the last gates of a circuit to the output. Therefore we use a *.vhd file file to specify the circuit. As long as the interconnects between two gates are 0 (all our test circuits had this property), using the Verilog file should not affect the results. If the Verilog file shall be used, the template file for the simulation has to be slightly adapted.

- *.sdf: The sdf file contains timing information, required for both delay models (ModelSim, Involution). It contains information about the timing regarding INTERCONNECTs and also about cell internal timing.

Configuration files

- Makefile: The Makefile includes the variables from the general configuration file (config.cfg) from the parent directory and its own config file (config.cfg). It also forwards all commands to the Sub-Makefile. Most of the required variables are already set in the config file in the circuits directory. Probably the most important thing about the Makefile is that all commands from the Sub-Makefile can be overridden. This can be used if a circuit requires special simulations, for example, which cannot be done with the default set of functions provided by the Involution Tool.

- generate.json, see Section A.2.1
- matching: Since the signals (especially the intermediate signals) can have different names between the *.sp file and the *.v file, we need a matching between their names. The left name is the name of the signal in the *.sp file, the right name the one from the *.v file. The following code snippet contains an example matching file:

```
din din
xmycir.g11:a temp1
xmycir.g12:a temp2
xmycir.g13:a temp3
xmycir.g14:a temp4
xmycir.g15:a temp5
xmycir.g16:a temp5
xmycir.g17:a temp51
xmycir.g18:a temp51
xmycir.g19:a temp52
xmycir.g20:a temp52
dout1 dout1
dout2 dout2
dout3 dout3
dout4 dout4
```

Listing A.7: Example configuration for matching file.

- report.cfg: Contains configuration for the automatic report generation, see Section A.2.3.
- config.cfg: Contains variables pointing to the configuration files and some other circuit specific staff which can not be specified globally in default config.cfg. The following code snippet shows an example configuration:

```
export TOP_DIR:=$(dir $(abspath $(lastword
↪  $(MAKEFILE_LIST))))
export INPUT_NAMES=din
ifndef MULTI_EXEC
include ../config.cfg
else
include ${ME_CIRCUIT_DIR}/config.cfg
endif

```

```
# figure prefix
export START_OUT_NAME=inv_tree_

# .do and .scr file configuration
export CIRCUIT_NAME=circuit_tb
export UNIT_NAME=c1
export VERILOG_FILE=inv_tree.v
export VCD_SIGNALS=din temp* dout*
export SDF_FILE=inv_tree_30.sdf
export REQUIRED_GATES=CKINVM1N
```

Listing A.8: Example configuration for circuit configuration file.

Specifying the *TOP_DIR* is required for most of the other path variables, since most of the path variables are defined relative from top dir (which is the directory of the current circuit). It is also necessary to specify the signal names, because they are used on various occasions. The switch *MULTI_EXEC* is necessary, because the config files are included in a different order if the circuit is executed from multi_exec tool (see Section A.4.1). The *START_OUT_NAME* is used for defining the prefix of the figure names. The next variables are used for generating the scripts which are executed by ModelSim, DesignCompiler and PrimeTime. *REQUIRED_GATES* contains a list of gates which are used by this circuit, and should be auto generated, more information in Section A.2.2

- schematic.png: Optional, is used for reporting (see Section A.3.2).

A.3.2 Generate schematic

How to create schematic from verilog file:

- Start *design_vision*
- File → Setup: Set search path for libraries → Apply
- File → Analyze → add vhdl File
- File → Elaborate
- Schematic → New Schematic View
- View → Save Screenshot As → Grab Screenshot of active view only
- Paint.NET invert colors (optional, because the background of the schematic is black, and this is probably not optimal for printing the report)

A.4 Tools

A.4.1 Multi exec tool

The multi execution tool can be used to simulate a circuit multiple times, with different configurations. Currently the waveform generation settings and some gate settings can be changed (T_P, exp_channel_location). The tool also allows to simulate the same circuit multiple times with the same configuration but with different randomly generated waveforms.

Configuration

The configuration of the multi_exec tool is contained in two different files:

multi_exec.cfg

```
export MULTI_EXEC:=1
export ME_CIRCUIT_DIR:=../../circuits/
export ME_CIRCUIT_UNDER_TEST:=${ME_CIRCUIT_DIR}/c17_slack
export ME_CONFIG_FILE:=multi_exec.json
export PRINT_LEVEL:=WARNING #INFORMATION, WARNING, FAIL,
↪ NONE
```

Listing A.9: Configuration for multi execution tool.

- MULTI_EXEC: the flag has to be set to 1, it is used on various occasions in the sub-makefiles.
- ME_CIRCUIT_DIR: path to the circuit directory containing all the different circuits that can be simulated.
- ME_CIRCUIT_UNDER_TEST: path to the circuit that should be simulated.
- ME_CONFIG_FILE: contains information about the different configurations that should be simulated, more information below.
- PRINT_LEVEL: can be useful for reducing the output of the Involution Tool to a minimum, otherwise the output can be quite verbose.

multi_exec.json

```
{
    "N": 2,
        "keep_waveform" : "True",
    "gate_generation": {
        "exp_channel_location_list": [
            "INPUT",
                "OUTPUT"
            ],
        "t_p_list": [
                0.8,
                1.2
            ]
        },
        "waveform_generation": [
            {
            "calc_next_transition_mode": 0,
                "groups": [
                ],
                "mue": 0.029,
                "sigma": 0.01
            },
            {
                "calc_next_transition_mode": 1,
                "mue": 0.029,
                "sigma": 0.02
            }
        ]
}
```

Listing A.10: Configuration for simulation runs of the multi exec tool.

- N: the number of simulations that should be made with a certain configuration.
- keep_waveform: If possible, keep the waveform between two simulation runs. If between two runs only parameters change which do not influence the waveform generation (T_P, exp_channel_location), the waveform can be kept. Therefore the tool always sweeps through those "waveform"-independent parameters first, and then through parameters which affect the waveform. If keep_waveform is true, the results of two simulation runs can be better compared, because otherwise two simulation runs would have a different input. Enabling this option also saves time, because the part which does not change (HSPICE Simulation, ModelSim simulation) is not re-executed, as it leads to the same result as the previous execution.
- gate_generation: We can sweep over the exp_channel_location and T_P, the pure delay in ps. These settings are used for all gates, so if multiple kinds of gates are used in the circuit, all gates get the same parameters.
- waveform_generation: Contains a list of waveform generation configurations (see Section A.2.1). If parameters are not specified here, the value of the circuit configuration file is used as fallback value.

Execution

The multi_exec tool can be executed with a Makefile. Currently the Makefile contains four recipes (*all*, *sim*, *report*, *clear*). *all* executes the simulation (*sim*) and afterwards the multi rerport is generated (*report*). For better understanding which configurations are generated by the tool, the tool saves the configurations in the temp folder under the name generate.json*num* and gate_config.json*num*. This way, the user can check if the configuration is as expected.

The reports of the different runs can be found in the results folder of the circuit, in a separate folder, where all reports of this execution are located.

Multi Reporting

The Involution Tool is also able to generate a report which summarizes the results of the simulation runs. The first step of the reporting tool is to combine the information of the results.json file. The configuration for the reporting is made in the multi_exec.cfg file:

```
export ME_COPY_PROPERTIES:=SPICEVERSION; SPICE_DCVersion;
↪  SPICE_DCDesign;ENV.*;
export ME_CALC_PROPERTIES:=SPICEpwr.*; .*Total_Total$$;
↪  .*total power$$; avgper.*; .*peak power; max_tc_.*;
↪  total_sum_error_.*
```

Listing A.11: Configuration for the multi exec reporting.

- ME_COPY_PROPERTIES: properties which stay the same throughout the execution can be specified here and are copied into the results.json file of the multi_report.
- ME_CALC_PROPERTIES: Numeric properties, which should be aggregated can be specified here. The reporting tool calculates the minimum / maximum / average value of the properties over all simulation runs.

For further processing of the data, the reporting tool also provides a csv export, which can be configured as follows:

```
export ME_CSV_PROPERTY_ORDER:=me_config_id, folder_name;
↪  SPICEpwr.*; .*Total_Total$$; .*total power$$; avgper.*;
↪  .*peak power; max_tc_.*; total_sum_error_.*
export ME_CSV_EXPORT_ALL_PROPERTIES:=True
export ME_CSV_ESCAPE_EQUAL_SIGN:=True
```

Listing A.12: Configuration for CSV export during multi exec reporting.

- ME_CSV_PROPERTY_ORDER: Defines the order of the columns, “important” columns can be placed at the beginning. This property must not to be specified. If not specified, all properties are added in an alphabetical order. Default: ""
- ME_CSV_EXPORT_ALL_PROPERTIES: True / False: adds all properties which are not specified in ME_CSV_PROPERTY_ORDER at the end in alphabetical order. Default: True
- ME_CSV_ESCAPE_EQUAL_SIGN: True / False: Escapes the equal sign "=", required for Excel, otherwise for example =-verbose shows an error, because it is interpreted as a formula. Default: False

The layout of the summary report can also be configured. The template for the report is placed in report_multi.tex. The report can consist of several sections:

- Basic information: Similar to the basic information section of the “single”-report.
- Configurations: All configurations used are listed in this section, and also all simulation runs which use that simulation are linked here. By clicking on one of the folder names, the corresponding “single”-report is opened (NOTE: Does not work with all types of pdf-readers).
- Power consumption: Contains a table for the average power consumption and also tables showing the minimum / maximum / average deviation of the power consumption compared to the SPICE simulation result (HSPICE) and to the result of the simulation of the SPICE trace with the same tool (column).
- Waveform comparison: Contains a table comparing the maximum relative and maximum absolute deviation for the transition count. Also shows the area of the deviation trace.
- Rankings: In this section, rankings for certain properties can be generated. Each table shows the specified value for each simulation run in an ordered way. This helps comparing the results of different configurations and different waveforms. Probably the csv export is an easier way to do this. For which properties a ranking table should be created can be specified in the variable “ME_RANKING_PROPERTIES” in the multi_exec.cfg file.

APPENDIX B

Involution channel implementation

B.1 Package

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

PACKAGE exp_channel_pkg IS
  COMPONENT exp_channel IS

    GENERIC (D_UP, D_DO, T_P : TIME);
    PORT (
      input : IN std_logic;
      output : OUT std_logic
    );

  END COMPONENT exp_channel;
END exp_channel_pkg;
```

Listing B.1: Package of the involution channel implementation.

B.2 Entity and architecture

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY exp_channel IS
  GENERIC (D_UP, D_DO, T_P : time);
  PORT ( input : IN std_logic;
   output : OUT std_logic);

END exp_channel;

------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.math_real.ALL;

ARCHITECTURE beh OF exp_channel IS
  -- as soon as the involution channel supports a
  ↪  non-symmetric threshold voltage
  -- remove this constants and replace with generics
  CONSTANT vdd : real := 1.0;
  CONSTANT vth : real := 0.5;
  CONSTANT tau_up: time := integer ( real ( (D_UP - T_P) /
  ↪  1 fs ) / (- LOG(1.0 - vth / vdd)) ) * 1 fs;
  CONSTANT tau_do: time := integer ( real ( (D_DO - T_P) /
  ↪  1 fs ) / (- LOG(vth / vdd)) ) * 1 fs;

  CONSTANT relTime: time := 1 fs;

BEGIN

  --###################################################

  exp_channel_involution: PROCESS (input)
    VARIABLE last_output_time : time := -1 sec;
    VARIABLE T, delay : time;
  BEGIN

  -- report "input transition at " & time'IMAGE(now);
-- report "last output at " & time'IMAGE(last_output_time);
  T := now - last_output_time;
-- report "T = " & time'IMAGE(T);

-- in VITAL they check for A'LAST_VALUE
-- but there also 'L' and 'H' of importance
IF rising_edge(input) THEN

  -- report "got rising edge";

  delay := D_UP + integer (
    real (tau_up / relTime) * LOG(
      1.0- EXP(
      -real( (T+D_DO)/relTime ) / real(tau_do/ relTime )
      )
    )
  ) * relTime;
  -- report "delay: " & time'IMAGE(delay);

  last_output_time := now + delay;

  IF (delay < 0 fs) THEN
    delay := 0 fs;
  END IF;
  output <= TRANSPORT '1' AFTER delay;

  ELSIF falling_edge(input) THEN
  -- report "got falling edge";

  delay := D_DO + integer (
    real (tau_do / relTime) * LOG(
      1.0- EXP(
      - real( (T+D_UP)/relTime ) / real(tau_up/ relTime )
      )
    )
  ) * relTime;
  -- report "delay: " & time'IMAGE(delay);

  last_output_time := now + delay;

  IF (delay < 0 fs) THEN
```

```
        delay := 0 fs;
      END IF;
      output <= TRANSPORT '0' AFTER delay;

      ELSIF (now = 0 fs) THEN
      output <= input;
      END IF;

    END PROCESS;

    --##################################################

END ARCHITECTURE;
```

Listing B.2: Entity and architecture of the involution channel implementation.

List of Figures

List of Tables

List of Algorithms

List of Listings

Glossary

AOI AND-OR-Invert (AOI) gates are constructed of one or more AND gates, followed by a NOR gate. The total number of used transistors is less than if each gate was implemented separately. 14

CSV comma separated value (CSV) is a file format for saving data in a tabular way. 17, 22

Design Compiler is a program from Synopsis, used for the synthesis of a HDL design. We use the Design Compiler especially for power estimation. 15, 16, 25, 31

HSPICE is an analog circuit simulator which can be used to simulate electrical circuits in stead-state, transient and frequency domains. 7, 11, 12, 15–21, 25–27, 29, 30

ModelSim is a HDL simulation environment for various HDL dialects by Mentor Graphics. xi, xiii, 1–3, 7, 8, 12–15, 17, 18, 20, 33, 63

NC-Sim simulation engine by Cadence for Verilog, VHDL and SystemC. 1, 4

OAI OR-AND-Invert (OAI) gates are constructed of one or more OR gates, followed by a NAND gate. The total number of used transistors is less than if each gate was implemented separately. 14

PrimeTime is a static timing analysis tool from Synopsis. It is used for timing, signal integrity, power and variation-aware analysis. We use PrimeTime especially for power estimation. 15, 16, 26, 30, 31

SAIF a switching activity information file is an ASCII based file format, which is used to store the switching activity of a design. The most important information it contains is the total duration when the net is a logic 0 (T0), logic 1 (T1), unknown (TX) and the total number of rise and fall transitions (TC). 15

SDF standard delay format (SDF) is used for representing and interpreting timing data at any stage during the design flow. We use the SDF file for specifying the interconnect delay and the delay in the cells. 12–14, 27

SPICE Simulation program with integrated circuit emphasis is a software which is used to simulate digital, analog and mixed electrical circuits. 1, 7, 11, 33

VCD a value change dump is an ASCII based file format. It is for example used by ModelSim to store the traces. It has a very compact structure. 12, 15–17

VCS simulation program by Synopsis for chip design. 1, 4

Verilog a hardware description language, used to model digital circuits. 7, 12, 13

VHDL very high speed integrated circuit hardware description language (VHDL) is a hardware description language, used to model digital circuits. 12, 13

Bibliography

[BJA00] 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, Apr 2000.

[BJV05] M. J. Bellido-Diaz, J. Juan-Chico, and M. Valencia. *Logic-Timing Simulation and the Degradation Delay Model.* Published by Imperial College Press and distriuted by Word Scientific Publishing Co., 2005.

[Cad15] Cadence Design Systems. Effective current source model (ECSM) timing and power specification. `http://projects.si2.org/openeda.si2.org/projects/omcdistrib/`, January 2015. Version 2.1.2, accessed 21-October-2018.

[FMN18] M. Függer, J. Maier, R. Najvirt, T. Nowak, and U. Schmid. A faithful binary circuit model with adversarial noise. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1327–1332, March 2018.

[FNN15] M. Függer, R. Najvirt, T. Nowak, and U. Schmid. Towards binary circuit models that faithfully capture physical solvability. In *Proceedings of the 2015 Design, Automation &#38; Test in Europe Conference &#38; Exhibition*, DATE '15, pages 1455–1460, San Jose, CA, USA, 2015. EDA Consortium.

[FNS13] M. Függer, T. Nowak, and U. Schmid. Unfaithful glitch propagation in existing binary circuit models. In *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*, pages 191–199, May 2013.

[FNS16] M. Függer, T. Nowak, and U. Schmid. Unfaithful glitch propagation in existing binary circuit models. *IEEE Transactions on Computers*, 65(3):964–978, March 2016.

[Loc16] D. Lockhart. Power estimation using synopsys primetime. `http://www.csl.cornell.edu/courses/ece5745/handouts/`

ece5745-tut5-pt.pdf, January 2016. Version 606ee8a, accessed 21-October-2018.

[Mar77] L. R. Marino. The effect of asynchronous inputs on sequential network reliability. *IEEE Transactions on Computers*, C-26(11):1082–1090, Nov 1977.

[Men16a] Mentor Graphics Corporation. *ModelSim Command Reference Manual*. Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777, 2016. Software Version 10.5c.

[Men16b] Mentor Graphics Corporation. *ModelSim User's Manual*. Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777, 2016. Software Version 10.5c.

[NSH15] R. Najvirt, U. Schmid, M. Hofbauer, M. Függer, T. Nowak, and K. Schweiger. Experimental validation of a faithful binary circuit model. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 355–360, New York, NY, USA, 2015. ACM.

[Ovi95] Open Verilog International. Standard delay format specification. http://ce.sharif.edu/courses/85-86/2/ce483/resources/root/sdf_3.0.pdf, May 1995. Version 3.0, accessed 21-October-2018.

[Syn10] Synopsis Inc. Power compiler user guide. http://eclass.uth.gr/eclass/modules/document/index.php?course=MHX303&download=/5346dc69nktr/5346dcb80FdV.pdf, June 2010. Version D-2010.03-SP2, accessed 21-October-2018.

[Syn13a] Synopsis Inc. *HSPICE Reference Manual: Commands and Control Options*, December 2013. Version I-2013.12.

[Syn13b] Synopsis Inc. *PrimeTime PX Tutorials*, June 2013. Version 000-1.

[Syn16] Synopsis Inc. CCS timing library characterization guidelines. https://usermanual.wiki/Document/ccstimingcharguidev34.1855048783.pdf, October 2016. Version 3.4, accessed 21-October-2018.

[Ung70] S. H. Unger. Asynchronous sequential switching circuits with unrestricted input changes. In *11th Annual Symposium on Switching and Automata Theory (SWAT1970)*, pages 114–121, Oct 1970.