HAL
archives-ouvertes.fr

# The Involution Tool for Accurate Digital Timingand Power Analysis

Daniel Ohlinger, Jürgen Maier, Matthias Függer, Ulrich Schmid

▶ **To cite this version:**

Daniel Ohlinger, Jürgen Maier, Matthias Függer, Ulrich Schmid. The Involution Tool for Accurate Digital Timingand Power Analysis. PATMOS 2019 - 29th International Symposium on Power and Timing Modeling, Optimization and Simulation, Jul 2019, Rhodes, Greece. 10.1109/PATMOS.2019.8862165 . hal-02395242

**HAL Id: hal-02395242**
**https://hal.inria.fr/hal-02395242**

Submitted on 5 Dec 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# The Involution Tool for Accurate Digital Timing and Power Analysis

Daniel Öhlinger*, Jürgen Maier*, Matthias Függer†, Ulrich Schmid*
*ECS Group, TU Wien
daniel.oehlinger@tuwien.ac.at, {jmaier, s}@ecs.tuwien.ac.at
†CNRS & LSV, ENS Paris-Saclay, Université Paris-Saclay & Inria
mfuegger@lsv.fr

***Abstract*—We introduce the prototype of a digital timing simulation and power analysis tool for integrated circuit (Involution Tool) which employs the involution delay model introduced by Függer et al. at DATE'15. Unlike the pure and inertial delay models typically used in digital timing analysis tools, the involution model faithfully captures pulse propagation. The presented tool is able to quantify for the first time the accuracy of the latter by facilitating comparisons of its timing and power predictions with both SPICE-generated results and results achieved by standard timing analysis tools. It is easily customizable, both w.r.t. different instances of the involution model and different circuits, and supports automatic test case generation, including parameter sweeping. We demonstrate its capabilities by providing timing and power analysis results for three circuits in varying technologies: an inverter tree, the clock tree of an open-source processor, and a combinational circuit that involves multi-input NAND gates. It turns out that the timing and power predictions of two natural types of involution models are significantly better than the predictions obtained by standard digital simulations for the inverter tree and the clock tree. For the NAND circuit, the performance is comparable but not significantly better. Our simulations thus confirm the benefits of the involution model, but also demonstrate shortcomings for multi-input gates.**

***Index Terms*—Digital timing simulation, design tools, delay models, pulse degradation, glitch propagation.**

## I. Introduction

Modern digital circuit design relies heavily on accurate timing analysis tools like Synopsys Prime Time, Mentor Questa, Cadence NC-Sim, or Synopsis VCS. They are able to accurately predict the signal propagation through a given circuit design, and thus identify setup/hold-violations and other timing-related problems in synchronous designs, for example. Moreover, they facilitate a reasonably accurate power analysis at early design stages [1], [2].

The "golden standard" here are fully-fledged analog simulations, e.g., using *SPICE* [3], which are based on detailed physical models of all elements in a digital standard-cell library. Since the simulation times of even moderately complex circuits are prohibitively excessive, however, digital timing analysis tools use *discrete-value* (typically binary) circuit models augmented by continuous-time delays. The latter is determined by elaborate timing prediction models like CCSM [4] and ECSM [5], which characterize the delay of a cell via (typically manufacturer-supplied) technology data and massive analog simulations. The gate and wire delay estimates obtained via CCSM or ECSM are then used for parametrizing pure or inertial delay channels [6] (e.g., in VHDL Vital or Verilog timing libraries). The resulting executable HDL simulation models are finally used in subsequent simulation and timing analysis runs.

This research was partially funded by the Austrian Science Fund (FWF) projects SIC (P26436) and RiSE (S11405), projects FREDDA (ANR-17-CE40-0013) and DEPEC MODE (Departement STIC), and by DigiCosme (working group HicDiesMeus).

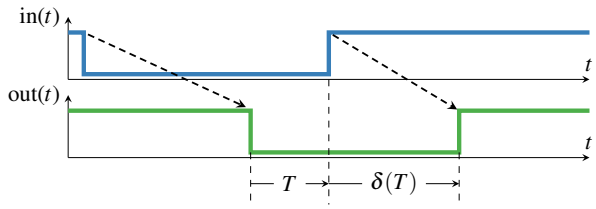



Fig. 1: Principle functionality of a single-history delay model. Based on the input-to-previous output transition time $T$, the delay $\delta(T)$ is determined.

Clearly, pre-computed delays are necessarily constants, i.e., remain the same throughout these runs. More accurate results can hence be expected from *dynamic* timing analysis techniques, which also consider signal trace-related effects. One example is pulse degradation, meaning that short input pulses usually get shorter when processed by a gate. The arguably simplest way to capture such dynamic effects are *single-history* channel models, which allow gate delays (modeled via the interconnecting channels) to vary dynamically in a trace. Single-history channels are characterized by a delay function $\delta$ that maps a transition occurring at the channel input at time $t$ to its corresponding output transition at time $t+\delta(T)$, where $T$ is the previous-output-to-input delay (cp. Fig. 1). If two succeeding input transitions would, according to $\delta(T)$, occur at the output in reversed order, they are said to cancel each other (shown in Fig. 2) and are removed. Note that single-history channels allow different rising and falling transition delays, specified by the delay functions $\delta_\uparrow$ and $\delta_\downarrow$, respectively.

The first proper single-history channel model was the *Degradation Delay Model (DDM)* introduced by Bellido-Díaz et al. [7], [8]. However, it was proven by Függer et al. in [9] that all existing delay models, including *DDM*, are not faithful: For the simple short-pulse filtration problem, it turned

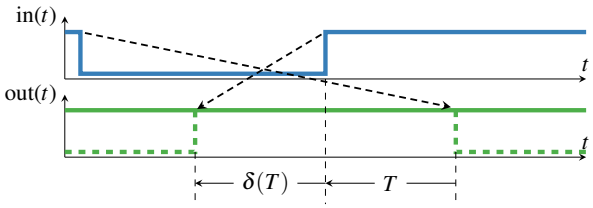



Fig. 2: The input pulse is so short that the transitions at the output appear in reverse order (dashed lines), i.e., cancel. Note that here $T < 0$ and $\delta(T) < 0$.

out that variants of it are solvable in physical implementations but not within existing circuit models, or vice versa. In [10], the authors therefore introduced the *involution model* (*IM*), which is the only delay model known so far that does not share this problem. Its distinguishing property is that its delay functions form involutions, i.e., are self-inverse, in the exact sense that $-\delta_\downarrow(-\delta_\uparrow(T)) = T$ and $-\delta_\uparrow(-\delta_\downarrow(T)) = T$. Types of the *IM* differ in the instance of delay functions they use. In [11], the authors utilized both *SPICE* simulations and real measurements to demonstrate that the involution model predicts the behavior of a real circuit, namely, an inverter chain, reasonably accurate.

*Main contributions:* As, to the best of our knowledge, there is no easy way to apply the involution model to custom signal traces, we developed the *Involution Tool* (`invTool`).[1] It allows to include and run any *IM* in state-of-the-art digital circuit simulation tools (e.g. *ModelSim*), and is embedded in a comprehensive test infrastructure that allows to generate user-controlled random input vectors, to run different analog/digital simulations, and to generate various reports on the results. Thanks to its ability to process the output of other simulation tools, in particular, *HSPICE*- or *Spectre*-generated traces, it easily allows to compare timing and power predictions of the involution model vs. other models. As switching from standard to *IM* simulation is essentially achieved by loading a different library, existing infrastructure, such as test scripts and input vectors, can be reused without modification.

In this paper, we provide the following contributions:

(i) We provide an overview of the features and some details of the implementation of the `invTool`. In particular, we briefly describe the two simple involution channels currently supported by our tool, namely, exp-channels based on simple exponential switching waveforms, and Hill-channels based on Hill functions that more closely match real switching waveforms.

(ii) We demonstrate the utility of the `invTool` by conducting a timing and power analysis of three example circuits: an inverter tree, the clock tree of an open source processor used in the tutorial [12], and a sample circuit that involves multi-input NAND gates, synthetized in two different technologies: 65 nm and 15 nm. It turns out that the timing and power predictions of both involution models are significantly better than the predictions obtained by standard digital simulations for the interter tree and the clock tree, albeit the Hill-channels surprisingly perform worse than the exp-channels for short pulses. For the NAND circuit, the predictions of the involution tool are comparable but not significantly better, sometimes even worse.

[1]The `invTool` can be found on GitHub: https://github.com/oehlinscher/InvolutionTool

Our experiments show that the *IM* is a viable approach for an accurate performance and power analysis of a circuit design. However, our detailed results also reveal some potential for improving the current approach, both w.r.t. its ability to accurately model short pulses and multi-input gates.

*Paper organization:* In Section II, we describe those features of the involution model that are instrumental for the `invTool`. Section III is devoted to an overview of how the *IM* is integrated into digital simulation tools. The architecture/implementation and the features provided by the `invTool` are described in Section IV, the experimental setup and the results obtained for our sample circuits are contained in Section V. Some conclusions in Section VI round-off our paper.

## II. Involution Model

In this section, we briefly summarize the most relevant properties of the involution model in Section II-A, and the general principle of circuit simulations in this model in Section II-B. The interested reader is referred to [10] for more details.

### A. Involution channels

When introducing the involution model in [10], Függer et al. have shown that its self-inverse delay functions arise naturally in a (generalized) standard analog model that consists of a pure delay component, a slew-rate limiter with generalized switching waveforms, and an ideal comparator, as shown in Fig. 3. First, the incoming, binary-valued input $u_i$ is delayed by a pure delay $T_p$, which is necessary to assure causal channels, i.e., $\delta_{\uparrow/\downarrow}(0) > 0$. For every transition on $u_d$, the generalized slew rate limiter immediately switches to the corresponding waveform ($f_\downarrow$ for a falling and $f_\uparrow$ for a rising transition) such that the value at $u_r$, representing the analog output voltage, does not jump. Finally, the comparator generates the output $u_o$ by discretizing the value of this waveform w.r.t. the threshold voltage $V_{th}$.

To calculate the delay function $\delta_\downarrow(T)$, as detailed in [10], one has to determine the value of $u_r$ as the falling transition on $u_d$ arrives and the time it takes from there onwards to return to $V_{th}$. For this purpose, we compute the delay of a perfectly idle channel $\delta^\uparrow_\infty = \lim_{T\to\infty} \delta_\uparrow(T)$ and $\delta^\downarrow_\infty = \lim_{T\to\infty} \delta_\downarrow(T)$ from a transition on $u_i$ to reaching $V_{th}$ on $u_r$ as

$$\delta^\uparrow_\infty = T_p + f^{-1}_\uparrow(V_{th}) \quad \text{and} \quad \delta^\downarrow_\infty = T_p + f^{-1}_\downarrow(V_{th}). \tag{1}$$

For a time difference of $T$ between the last transition on $u_0$ and the current one on $u_d$ the value of $u_r$ can now be expressed as $f_\uparrow(T + \delta^\uparrow_\infty)$. To finally get $\delta_\downarrow(T)$, the time it takes for $f_\downarrow$ to reach $u_r$ has to be subtracted from $\delta^\downarrow_\infty$, i.e.,

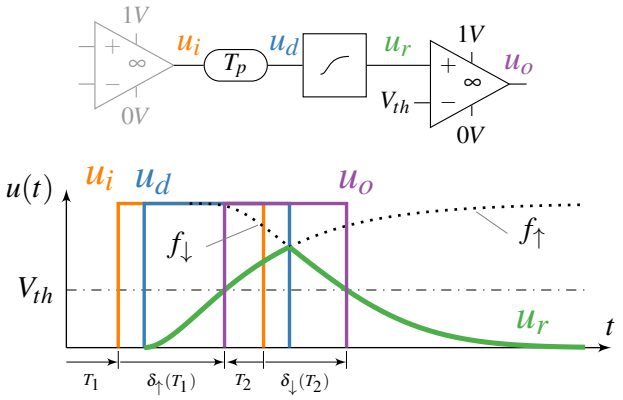



Fig. 3: Simple analog channel model (upper part) with a sample execution (bottom part) taken from [10]. At a transition on $u_d$ the transitions waveforms are immediately switched.

$$\begin{aligned}\delta_\uparrow(T) &= \delta_\infty^\uparrow - f_\uparrow^{-1}(f_\downarrow(T+\delta_\infty^\downarrow)) \quad \text{and}\\ \delta_\downarrow(T) &= \delta_\infty^\downarrow - f_\downarrow^{-1}(f_\uparrow(T+\delta_\infty^\uparrow)).\end{aligned} \tag{2}$$

If the slew rate limiter is implemented as a first-order RC low pass filter, for example, we obtain what will be called an *exp-channel* in the sequel: The switching waveforms are $f_\downarrow(t) = 1 - f_\uparrow(t) = e^{-t/\tau}$ here, with $\tau$ being the RC time constant that determines its steepness. Inserting these functions and their inverses into Eq. (2) and Eq. (1), we obtain

$$\begin{aligned}\delta_\uparrow(T) &= T_p - \tau\ln(1-V_{th}) + \tau\ln(1-e^{-(T+T_p-\tau\ln(V_{th}))/\tau})\\ \delta_\downarrow(T) &= T_p - \tau\ln(V_{th}) + \tau\ln(1-e^{-(T+T_p-\tau\ln(1-V_{th}))/\tau})\end{aligned}$$

Note that $T_p > 0$ is required to ensure causality of the exp-channel, i.e., $\delta(0) > 0$.

The `invTool` also supports involution channels based on the well-known *Hill function* [13], which matches real switching waveforms better than the exponential function; they will be called *Hill-channels* in the sequel. Their switching waveforms are $f_\downarrow(t) = 1 - f_\uparrow(t) = \frac{t^n}{k^n+t^n}$, the parameter $k$ basically determines when the threshold is reached and thus primarily depends on $\delta_\infty^\uparrow$ resp. $\delta_\infty^\downarrow$, $T_p$ and $V_{th}$. The parameter $n$ (the Hill coefficient) can be chosen freely to adjust the actual switching speed. By using

$$\delta_\infty^\uparrow = T_p + k_\uparrow \sqrt[n_\uparrow]{\frac{1-V_{th}}{V_{th}}} \quad \text{and} \quad \delta_\infty^\downarrow = T_p + k_\downarrow \sqrt[n_\downarrow]{\frac{V_{th}}{1-V_{th}}}$$

and again inserting these functions and their inverses into Eq. (2) we obtain

$$\begin{aligned}\delta_\uparrow(T) &= \delta_\infty^\uparrow - k_\uparrow\left(\frac{k_\downarrow}{T+\delta_\infty^\downarrow}\right)^{\frac{n_\downarrow}{n_\uparrow}},\\ \delta_\downarrow(T) &= \delta_\infty^\downarrow - k_\downarrow\left(\frac{k_\uparrow}{T+\delta_\infty^\uparrow}\right)^{\frac{n_\uparrow}{n_\downarrow}}.\end{aligned}$$

### B. Simulations

Viewed at the level of digital signals, the behavior of an involution channel is defined by the channel simulation Algorithm 1, which maps channel input signal $s$ (with event list Input) to channel output signal $f_C(s)$ (with event list Output). Event lists hold the transitions $(t, s)$ of a signal, where $t$ is the occurrence time and $s \in \{0, 1\}$ the signal value after the transition. Signals contain an initial transition at time $-\infty$, potentially followed by transitions with increasing non-negative times and alternating values. Since output signal transitions of an involution channel may cancel each other, the simulation algorithm also maintains an internal event list Pending: An output transition is only fixed, i.e., moved from Pending to Output, when it cannot be canceled later on.

**Algorithm 1** Channel algorithm, up to time $\tau$.

```
Pending ← []; Output ← []
(−∞, x_−∞) ← initial event in Input
add (−∞, x_−∞) to Output
Prev ← (−∞, x_−∞)
for (t, x) in Input with 0 ≤ t ≤ τ, ascending in time t do
    (t′, x′) ← Prev
    if x = 1 then δ ← δ↑(t − t′) else δ ← δ↓(t − t′) endif
    Prev ← (t + δ, x)
    if t + δ ≤ t′ then
        remove (t′, x′) from Pending
    else
        if exists move (t′, x′) from Pending to Output
        add (t + δ, x) to Pending
    end if
end for
if exists (t, x) in Pending with t ≤ τ, add it to Output
return Output
```

When being implemented in a digital simulation tool (in our case *ModelSim*), Algorithm 1 can be simplified dramatically, since transitions in the wrong temporal order are dropped automatically. Therefore, it suffices to calculate $\delta(T)$ and delay the input transition by exactly that amount, leaving it to *ModelSim* to cancel wrongly ordered transitions. Note that we did not verify this property for alternative simulation suites.

## III. Incorporating *IM* in ModelSim in Practice

One of the main goals for the development of the `invTool` was our desire to perform circuit simulations using the *IM* without the need to install and utilize some non-standard software tool. For that reason, we used VHDL Vital as our guidance for the development of the `invTool`. As a result, changing between our *IM* implementation and the former is achieved simply by switching libraries, which facilitates code and test setup re-using.

Consequently, our solution not only has the same structure as VHDL Vital, but also responds to the same variables and is also written in VHDL. Simulations in the `invTool` are completely controlled by *ModelSim*, which makes it possible to use all its features without restrictions: Based on the next input transition time at the channel input, the algorithm determines $T$ and the resulting $\delta(T)$, and adds the transition to

the channel's output. This is done separately for each channel, as their parameters can differ.

For simulations using the `invTool`, one hence needs exactly the same input files as for any standard post-layout simulation: the circuit, a testbench, and the timing characteristics stored in *.sdf* files. The latter contain the static delay of each gate ($\delta_\infty$) in the circuit and the interconnects in between. While VHDL Vital uses essentially pure/inertial delays with a priori given fixed delay values, the *IM* calculates the parameters for the delay functions $\delta_\uparrow$ and $\delta_\downarrow$ as introduced in Section II-A, using a user-defined value for the pure delay parameter $T_p$. For Hill-channels, the user can also set the values for $n_\uparrow$ and $n_\downarrow$, which are not determined by the delay characteristics stored in *.sdf* files. Note that neither $T_p$ nor $n_\uparrow$ and $n_\downarrow$ are always easy to guess, so we will experimentally determine their impact in Section V.

A crucial task for using the `invTool` in practice is to extend the set of available basic gates, which of course has to be done only once for a new gate. It essentially consists of modeling the Boolean functionality in VHDL and connecting in- and outputs via suitable *IM* channels. While single input-single output channels are easy to handle, things get more complicated for multi-input gates, as there are different possible locations for placing the *IM* channels.

## IV. Involution Tool (`invTool`)

Our Involution Tool `invTool`, which was originally developed in [14], is a complete framework for the systematic and automatic evaluation of different delay prediction methods for several power and timing metrics. It allows to generate user-controlled random input vectors, to run different analog/digital simulations, to automatically sweep the ranges of user-defined parameters, and to generate various reports on the results. In Section IV-A, we outline the workflow of using the `invTool`, which also provides a glimpse on its overall architecture and its core features. In Section IV-B, we explain how the tool supports parameter sweeping and multiple simulation runs.

### A. Workflow of the `invTool`

The overall information flow in our tool is shown in Fig. 4. Around the central delay estimation method (termed *Digital Simulation* in our figure, see Section III for details), which is implemented in *ModelSim* (also called QuestaSim), we developed a complete framework that handles everything from waveform generation to evaluation and reporting autonomously. The only required inputs are (i) a *SPICE*/Verilog description of the circuit under test as well as (ii) the corresponding timing file, which can be created using *Cadence Encounter* [12], for example. Due to its modular structure, each part of the toolchain can be substituted, provided that the interfaces do not change. Note that this feature became crucial in the course of the experiments described in this paper, as we encountered numerical issues with *HSPICE* that forced us to switch to *Spectre* for some circuits/technologies.

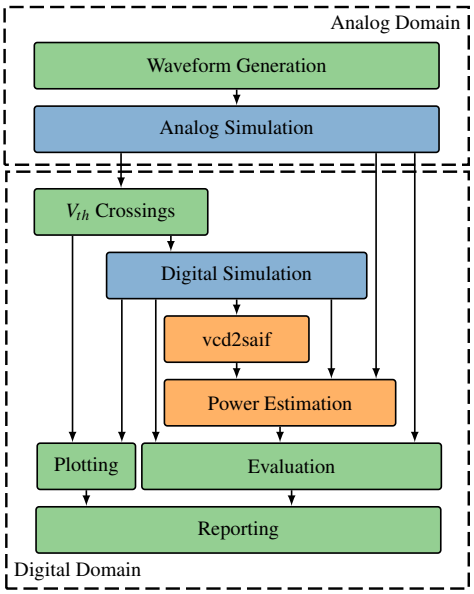



Fig. 4: Workflow in `invTool`. The green parts had to be implemented from scratch. The blue parts have been available, albeit the available resources had to be extended significantly. Orange parts could be used almost out of the box.

*Waveform Generation:* The first task of `invTool` is to generate the test stimuli for the circuit. Since suitable test vectors largely depend on the actual circuit, we resorted to randomly generating transitions on the input(s) here. More specifically, for each input, the time until the next transition follows a (lower bounded) Gaussian distribution, whose parameters ($\mu$, $\sigma$) can be set by the user. This way, desired constellations, such as pulse cancellations in the circuit, can be made more probable than others. The user can furthermore choose between two modes of operation: In one the randomly determined delay is added to the last global transition time, while in the other it is added to the last transition at the corresponding input. By grouping multiple inputs, the user can increase the probability for transitions in a short period of time for the signals of the group, which can be useful for multi-input gates. All these choices can be saved, together with the parameters ($\mu$, $\sigma$), in a configuration file, which is finally read by a Python script that ultimately determines the actual transition times.

*Analog Simulation:* To evaluate the accuracy of our predictions, we need a "golden reference", which are currently analog simulations using *HSPICE* or *Spectre*. Intially, the randomly generated input transition times are transformed to an analog curve using a piecewise linear (PWL) source with a rise/fall time of 1 ps. This source is then added to a template file, which also imports the circuit under test and is used for the subsequent analog simulations. Since the input transitions are very steep (mostly likely too steep for certain circuits), it is possible to employ inverter chains in front of each input to shape the signal. For the technologies used in our experiments, two inverters proved to be sufficient to

generate realistic signals. Note that we provided a separate supply voltage source to these shaping circuits to prevent any effect on our power measurements.

Our analog simulations provide us with both (i) a reference for the actual power consumption and (ii) reference switching traces (i.e., threshold crossings) at different nodes within the circuit. In the next step of our tool chain, these *SPICE* switching traces are fed to a Python script (termed $V_{th}$ *Crossings* in Fig. 4), which generates input files that can be interpreted by the succeeding digital simulation tool, in our case, *ModelSim*.

*Digital Simulation:* All digital simulations, as described in Section III, are performed by *ModelSim*. The input file, which is extracted from the analog simulation switching traces, are read by the testbench and applied to the circuit under test as inputs. Two different delay models can be used in these simulations: the built-in VHDL Vital library or our *IM* library, in the sequel denoted as `STA` and `INV`.

Of course, as pointed out in Section III, every gate used in the circuit under test has to be present in the respective library. In the case of *IM*, the `invTool` is capable of automatically generating a corresponding entry for simple gates, i.e., those consisting of a single combinational function only. The parameters of the gates (in particular, channel type, pure delay, location and specific parameters) can be specified in a configuration file. For more complex gates, the user has to enter the description of the gate in VHDL manually, which gives full control on channel types and locations.

*Power Estimation:* In addition to the *SPICE*-generated analog power estimation, the `invTool` allows to use multiple digital power estimation tools, currently *Design Compiler* and *PrimeTime*, where for the latter two different modes (average and time-based) can be chosen (which can differ quite significantly). As the reference power estimation, the `invTool` allows to choose between the following two alternatives: (i) the *SPICE* analog power estimation and (ii) the *Design Compiler* and *PrimeTime* power estimation generated for the *SPICE* switching traces (obtained from the *SPICE* waveforms). For power comparison, the results of the *ModelSim* simulation (`STA`, `INV`) are fed into the the same tools as in (ii) and compared to the reference value. Note that the *Design Compiler* and the average-based simulation mode of *PrimeTime* only use the switching activity information file (.saif), whereas the time based simulation is based on a value change dump file (.vcd), which also contains information about the time of the transitions.

*Evaluation:* As a first step, the tool converts all the results from the previous stages into a unified format. The following four metrics are supported:

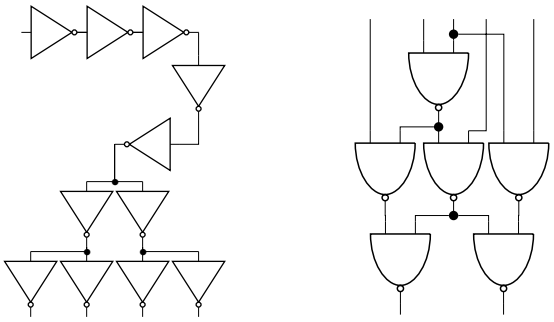
- *Power deviation*: As already mentioned, the results of the digital simulation of the involution gates (`INV`) and the standard gates (`STA`) are compared to the two types of reference values (where the second type actually produces three values, one for each tool and option). This way, we can identify deviations and bias caused by the different power estimation techniques.
- The *number of transitions* in the digital simulation trace, for the signal at every node, is calculated and compared to the corresponding *SPICE* switching traces. Both the average deviation and the maximum deviation are computed.
- The deviation between a digital simulation trace and the corresponding *SPICE* switching trace is measured via the *area under the deviation trace*, which can be computed with and without induced/suppressed glitches. Note that the `invTool` actually computes signed areas, normalized to $V_{DD} = 1$ and per transition, with the sign depending on whether the transition of the reference signal comes first (negative, representing a *trailing* transition) or not (positive, representing a *leading* transition). Note that the normalized area effectively represents the (average) time a transition happens before or after the reference signal. This feature is extremely useful for determining a bias in the delays, e.g., caused by inaccurate information in the *.sdf* file.
- During the comparison of a *SPICE* switching trace and the corresponding digital simulation trace, the tool checks whether a pulse (= two subsequent transitions, starting from and returning to the current level of the other trace) happens in one trace without any transition in the other one: If such a pulse occurs in the *SPICE* switching trace, we call it an *original suppressed glitch*, otherwise an *original induced glitch*. The `invTool` also evaluates whether a pulse in one trace properly contains a pulse in the other trace; we call such a pulse an *inverted suppressed glitch* resp. an *inverted induced glitch*. It outputs the number of those glitches divided by the total number of transitions in the corresponding signal.

Section V will show that these relatively simple measures provide useful information on the influence of certain model parameters.

*Reporting:* The final step of the toolchain allows to automatically generate a LaTeX report. The default report shows (i) information about the simulation environment, (ii) waveform generation settings, (iii) power consumption, (iv) trace comparison results, (v) plots and (vi) the schematic of the circuit. Detailed information about the trace comparison results, for every node, is also stored in a .csv file. The report can hence easily be customized by the user: format and content can be configured by means of a template, which allows to incorporate all values extracted and calculated during the evaluation.

### B. Multi-execution and parameter sweeping

The `invTool` is capable of automatically performing multiple simulation runs, by invoking the toolchain several times. Since the waveforms are generated randomly, this is an important feature for obtaining reasonable results. Furthermore, sweeping over different channel parameters and waveform generation settings is supported by the tool. In the following, we denote a complete sweep over all configured parameters as



(a) inverter tree (b) NAND circuit

Fig. 5: Schematic of circuits used for simulation.

*simulation run*, whereas *simulation* denotes a single execution with a certain configuration.

The following channel parameters are supported:

- *Pure delay* $T_p$.
- *Channel location*: Decides whether the channels are placed at the inputs or the outputs of the gates.
- *Channel-specific parameters*: For the Hill-channels, $n_\uparrow$ and $n_\downarrow$ can be specified.

An important feature of our multi-execution is that the generated waveforms can be retained between subsequent simulations, as long as only the channel parameters are modified. Since these parameters only influence the involution channel parts of the simulation, which are solely handled by *ModelSim* a lot of simulation time (up to 2/3) can be saved. After all, the analog simulation needed for waveform generation is the most time-consuming part of the toolchain. Moreover, starting from the same input waveforms increases reproducability and comparability of the simulation results.

After finishing all simulation runs in multi-execution, the tool aggregates (i.e., averages) the results from all simulations and generates a report, again based on a LaTeX template. Moreover, the results are exported to a .csv file again, which allows further post-processing.

## V. Results

To demonstrate the utility of the `invTool` and to validate/reject some conjectures w.r.t. the existing involution model, we present the results of the evaluation of three different circuits, namely, an inverter tree, where we used a standard 65 nm UMC library ($V_{DD} = 1.1\,\mathrm{V}$) that was simulated using *SPICE*, the clock tree of an open-source MIPS processor [12], and a custom NAND circuit. For the latter two circuits, we used the 15 nm Nangate Open Cell Library with FreePDK15™ FinFET models [15] ($V_{DD} = 0.8\,\mathrm{V}$). Note that we had to resort to *Spectre* for this technology, as we ran into numerical issues with *SPICE*; due to the modular design of the `invTool`, this switch of tools was easily achievable. Below, we describe the results of the evaluation of these circuits and draw some conclusions from our findings.

The first thing to note is that the tool-generated delay estimation files (*.sdf*) turned out to be very inaccurate: they led to significant deviations between analog and digital simulations already for very broad pulses (i.e., large $\mu$). Therefore, we eventually decided to create custom delay files based on our analog simulations. By doing so, we achieved precise delay predictions,[2] with an accuracy of $\pm 1\,\mathrm{ps}$. These custom delay files also serve another purpose, namely incorporating the interconnect delay in the gate delay. This is necessary, because the model does not support interconnect delay yet.

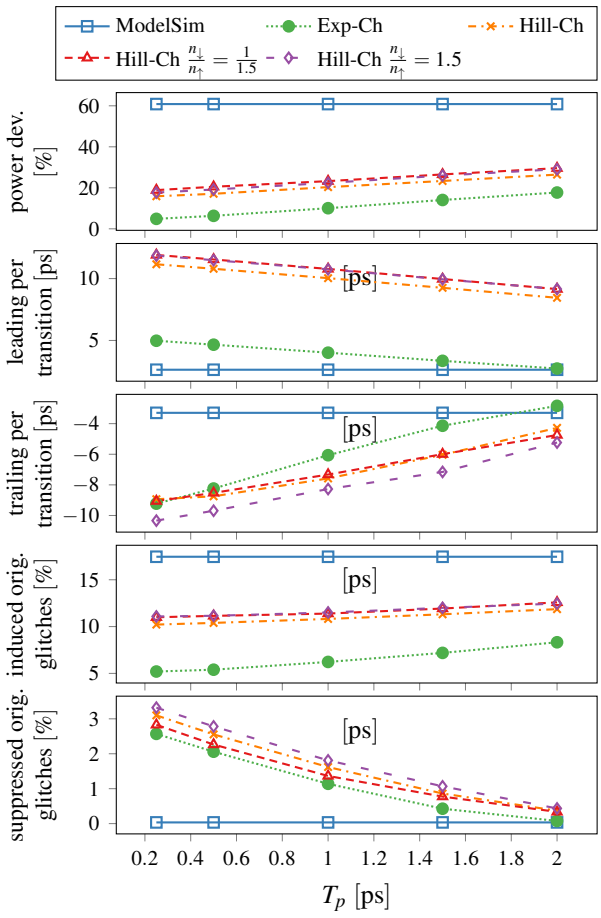



Fig. 6: Inverter tree simulation results (65 nm) for $\mu = 29\,\mathrm{ps}$.

Among all the metrics extractable by the `invTool`, as described in Section IV-A, we selected only (i) power deviation, (ii) leading and trailing normalized area under the deviation trace (without glitches), and (iii) original induced and suppressed glitches. In all simulations, we used both exp-channels and Hill-channels and the default *ModelSim* delay model as a baseline. Using the multi-execution feature, we swept over $T_p$ and, for the Hill-channels, also varied the quotient $\frac{n_\downarrow}{n_\uparrow}$.

### A. Inverter tree

Fig. 5a shows the inverter tree used for our simulation experiments and Fig. 6 the results for $\mu = 29\,\mathrm{ps}$. Note that

[2] As a consequence, the overall area under the deviation trace has been significantly decreased. For example, for $\mu = 100ps$ (the case shown in Fig. 7), from over 30 to below 3.5.

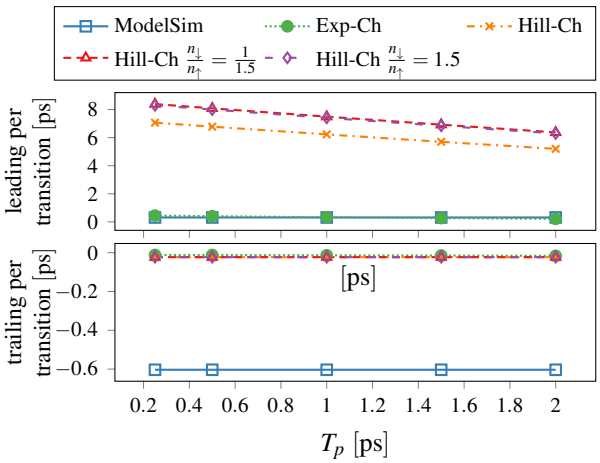



Fig. 7: Inverter tree simulation results (65 nm) for $\mu = 100$ ps.

$\delta^\downarrow_\infty, \delta^\uparrow_\infty$ can be as low as 7.7 ps. in the 65 nm technology used here.

With respect to the power estimation accuracy,[3] the exp-channel clearly performs best, immediately followed by the Hill-channels. The increase in the power deviation with increasing $T_p$ is a result of the higher amount of induced glitches resp. the lower amount of suppressed glitches: This is actually a consequence of the fact that $\delta^\uparrow_\infty$ and $\delta^\downarrow_\infty$ are kept constant while increasing the pure delay, which effectively results in steeper switching waveforms and less cancellations. The former also explains why trailing (= trailing normalized area under the deviation trace) decreases with $T_p$ for all involutions. Interestingly, leading also decreases in the same fashion, albeit considerably less, which seems to contradict this explanation. However, increasing $T_p$ of the overall delay *reduces* the latter's dependence on the switching waveform. So, on the one hand, increasing the pure delay decreases leading, whereas steeper switching waveforms increase it (see below for a possible explanation of this phenomenon in general).

Overall, except for suppressed glitches, the default VHDL Vital delay model performs poorly compared to any involution model, in particular, relative to the exp-channel. Note that this is even true w.r.t. leading and trailing if one also counts induced glitches (not shown in Fig. 6).

A surprising finding is that Hill-channels, which are based on waveforms that match real switching waveforms better than the exponential functions used for the exp-channel (see Section II-A), provide considerably worse delay estimations: this can be deduced from the large leading in Fig. 6. We conjecture that this is a consequence of the instantaneous switching between up- and down waveforms, which underlies the current involution model: As the Hill functions are very steep near $V_{th}$, the corresponding *IM* underestimates the delays.

For broader pulses ($\mu = 100$ ps), we found that power is estimated accurately and no glitches are induced or suppressed. As revealed by Fig. 7, however, the Hill-channel still substantially underestimates delays, whereas both the exp-channel and the *ModelSim* delay model perform reasonably well. With respect to trailing, involutions now always outperform *ModelSim*, however.

[3]Note that we did not use a *.spef* file for the power estimation with *Design Compiler* and *PrimeTime* when using the 65 nm library.

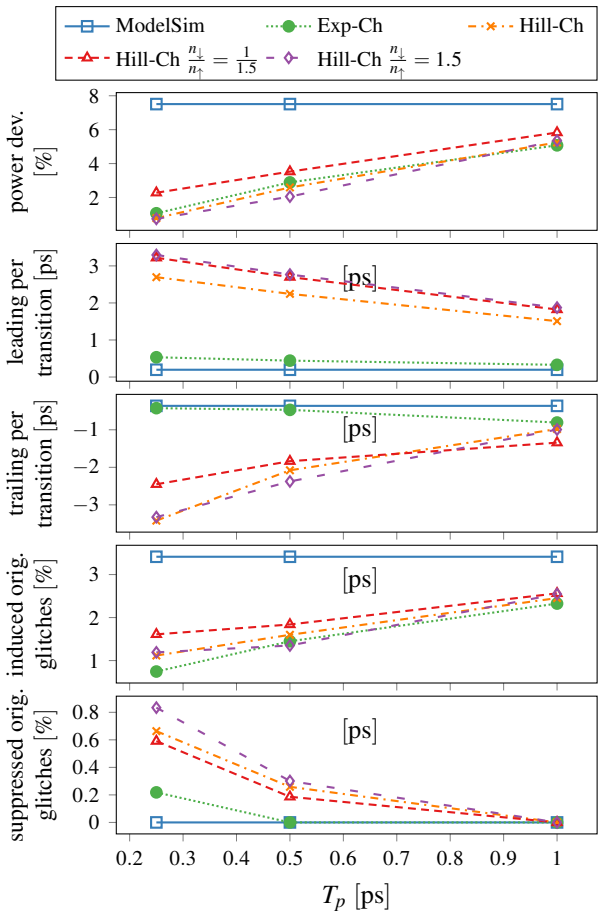



Fig. 8: MIPS clock circuit simulation results (15 nm) for $\mu = 29$ ps.

## B. MIPS clock

The clocktree, synthesized in 15 nm technology, comprises of 227 inverters which drive 123 Flipflops. Fig. 8 shows the results of our evaluation of the MIPS clock tree. Since $\delta^\downarrow_\infty, \delta^\uparrow_\infty$ can be as low as 1.2 ps here, we had to restrict the range for $T_p$ appropriately. Nevertheless, qualitatively, the results are quite similar to the ones for the inverter tree (Fig. 6 resp. Fig. 7). Note that $\mu = 29$ ps for our faster 15 nm technology is more like $\mu = 100$ ps for the slower 65 nm technology, however.

## C. NAND circuit

For synthesizing the NAND circuit shown in Fig. 5b, we also used our 15 nm technology, which resulted in $\delta^\downarrow_\infty, \delta^\uparrow_\infty$ being as low as 2.7 ps. We placed the channels either at each input or at the output of a gate, and observed that the latter yields better results. Moreover, if the channels are at the inputs,

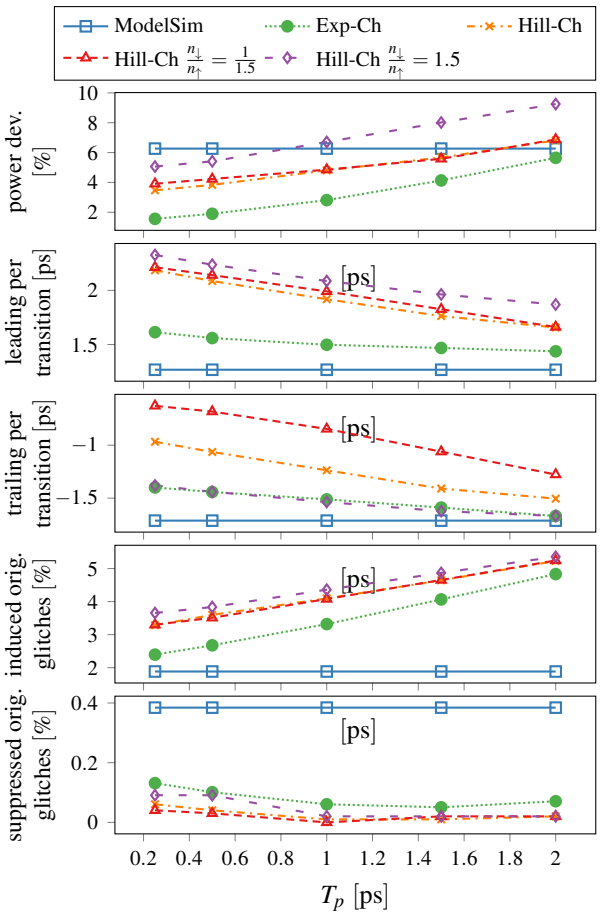



Fig. 9: NAND circuit simulation results (15 nm) for $\mu = 29$ ps, with channels placed at the output.

a Hill-channel with a quotient of $\frac{n_\downarrow}{n_\uparrow} = 1.5$ performs best, whereas for channels at the outputs the standard Hill-channel with $\frac{n_\downarrow}{n_\uparrow} = 1$ wins. Like in the inverter tree, exp-channels generally outperform Hill-channels w.r.t. leading and trailing.

However, it is apparent from the results in Fig. 9 that the involution models do not outperform the default *ModelSim* delay model as significantly as for the inverter tree. Two obvious reasons for this fact are the faster technology and also logical masking, which makes the NAND circuit less susceptible to propagating glitches. In addition, however, we also conjecture that the existing involution channels have deficiencies in accurately modeling multi-input gates in general. These deficiencies arise, since the involution channel is either placed at each input or at the output, and does not consider all inputs together.

## VI. Conclusions

In this paper, we presented an overview of the features and the internal architecture of the Involution Tool, a custom simulation environment for the involution delay model. Thanks to its embedding into the state-of-the-art digital simulation tool *ModelSim*, and its compatibility with other tools like *SPICE*, exisiting circuits can be easily simulated in the involution model and its performance compared to other prediction methods. Complemented by automatic waveform generation, parameter sweeping capabilities and automatic report generation facilities, it allows the systematic experimental evaluation of different circuits in different model variants.

We demonstrated its capabilities by means of analyzing several different circuits, in different technologies. Whereas our experiments confirmed the superiority of the involution model in general, they also revealed two unexpected facts. First, it turned out that Hill-channels, which are based on more realistic switchings waveforms, provide worse delay predictions than the simple exp-channels for short pulses. Second, the considerably less superior predictions of the involution model for our NAND circuit suggest that the existing involution channels are not fully adequate for accurately modeling multi-input gates.

## References


[1] F. Najm, "A survey of power estimation techniques in VLSI circuits," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 446–455, 1994.

[2] M. Favalli and L. Benini, "Analysis of glitch power dissipation in CMOS ICs," in *Proceedings of the 1995 international symposium on Low power design*, ser. ISLPED '95. New York, NY, USA: ACM, 1995, pp. 123–128.

[3] L. W. Nagel and D. Pederson, "SPICE (Simulation Program with Integrated Circuit Emphasis)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, 1973.

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

[5] *Effective Current Source Model (ECSM) Timing and Power Specification*, Cadence Design Systems, January 2015, version 2.1.2.

[6] S. H. Unger, "Asynchronous sequential switching circuits with unrestricted input changes," *IEEE Transaction on Computers*, vol. 20, no. 12, pp. 1437–1444, 1971.

[7] M. J. Bellido-Díaz, 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*, vol. 147, no. 2, pp. 107–117, 2000.

[8] M. J. Bellido-Díaz, J. Juan-Chico, and M. Valencia, *Logic-Timing Simulation and the Degradation Delay Model*. London: Imperial College Press, 2006.

[9] M. Függer, T. Nowak, and U. Schmid, "Unfaithful glitch propagation in existing binary circuit models," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 964–978, March 2016.

[10] 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 & Test in Europe Conference & Exhibition*, ser. DATE'15. San Jose, CA, USA: EDA Consortium, 2015, pp. 1455–1460.

[11] 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*, ser. GLSVLSI'15. New York, NY, USA: ACM, 2015, pp. 355–360.

[12] J. C. Ferreira, "Physical synthesis with encounter (cadence)," https://paginas.fe.up.pt/~jcf/ensino/disciplinas/mieec/pcvlsi/2015-16/tut_encounter/tut_encounter.html, 2015/16.

[13] A. V. Hill, "The possible effects of the aggregation of the molecules of haemoglobin on its dissociation curves," *Journal of Physiology (London)*, vol. 40, pp. 4–7, 1910.

[14] D. Öhlinger, "Involution tool," E191 - Institut für Computer Engineering; Technische Universität Wien, Tech. Rep. TUW-278633, 2018. [Online]. Available: https://publik.tuwien.ac.at/files/publik_278633.pdf

[15] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ser. ISPD '15. New York, NY, USA: ACM, 2015, pp. 171–178.