Involution Tool Manual

Daniel Öhlinger

May 10, 2020

Contents

Version history

Date	Revision	Author	Changes
12/19/2018	1.0	D. Öhlinger	Initial version, based on the bachelor thesis [1]
05/01/2019	2.0	D. Öhlinger	Modified structure for newly added features during PATMOS paper
05/10/2020	2.1	D. Öhlinger	Added documentation TODO section

1 Introduction

The Involution Tool (`invTool`) is a framework for comparing different digital delay prediction methods, especially the involution delay model and the standard *ModelSim* model, with an analog *SPICE* reference. The results are compared in terms of several metrics, see Section 6. Since the input waveforms are generated randomly, and several parameters can be adjusted for the involution delay model, the `invTool` also offers an opportunity to invoke the toolchain several times (see Section 5.1.2), in order to obtain meaningful results and find the best fitting parameters.

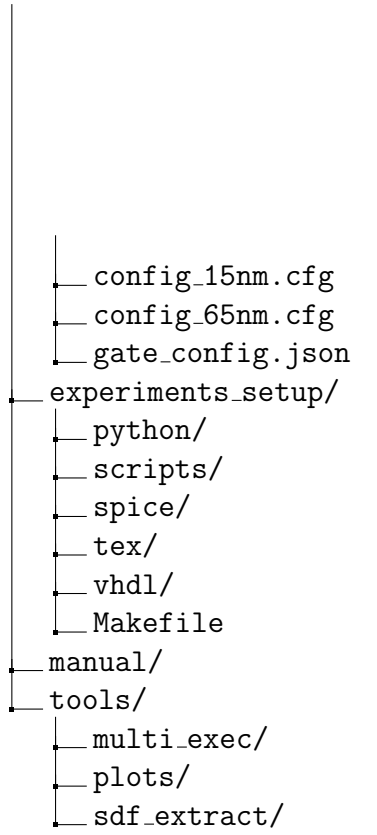
2 General

In this Section, the overall structure of the Involution Tool (`invTool`) is presented and the workflow is explained. It is explained how the parts of the `invTool` interact with each other.

2.1 Structure of the Involution Tool

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

```
circuits/
├── c17_slack_15nm/
├── c17_slack_65nm/
├── inv_tree_15nm/
├── inv_tree_65nm/
├── mips_clock_15nm/
└── config.cfg
```



2.1.1 Circuits

This is the folder where new circuits should be added. *c17_slack_15nm*, *c17_slack_65nm*, *inv_tree_15nm*, *inv_tree_65nm*, and *mips_clock_15nm* are five 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, independent of the technology. These variables can be overridden in the *config.cfg* file of each circuit, which also includes the default *config_xxnm.cfg* file, depending on the used technology. More information can be found in Section 4.1. Most of the variables in the *config.cfg* files are used for defining the folder structure of a specific circuit and the libraries to use.

The recommended (and default) structure for a circuit is:

- input: contains *main_new_exp.sp*, a *SPICE* file where the placeholders (for example for the input traces) are already replaced, and the generated waveform saved in a **.json* file.
- spice: contains the trace and logfiles of the *SPICE* simulation.
- crossings: contains a **.json* file with all the extracted crossings from either the *HSPICE* trace (**.tr0*) file or the value change dump (**.vcd*) file obtained with *Spectre*.

- 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 log files, and especially the **.vcd* files, which are later used for power estimation and plotting.
- power: contains the log files 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 *SPICE* 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. The waveform uses as input is also saved, together with the used configurations, so that the results can be reproduced.
- gates: simple gates can be created automatically (more information in Section 3.2). 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 4.1.

2.1.2 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 2.2.

- python: contains python scripts used for various steps throughout the whole workflow (generating waveforms, parsing trace files, plotting figures and 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 3.3.
- spice: contains useful snippets which can be added to the **.sp* file of the circuit. Currently only contains files for shaping for the 15 nm (*shaping_15nm.sp*) and 65 nm (*shaping_65nm.sp*) technology can be used for adding an inverter chain at each input.
- vhdl: contains the involution delay channel implementations (exp-channel, Hill-channel), 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.

2.1.3 Manual

Contains the LaTeX sources for this manual, and also the resulting manual.pdf.

2.1.4 Tools

This folder contains tools which can be used in combination with the Involution Tool. More information in Section 5.

2.2 Workflow of the Involution Tool

The flowchart in Figure 1 shows the workflow of the `invTool`. Note that orange parts could be used almost out of the box, whereas blue parts needed significant extension. Green parts are completely implemented from scratch.

The `invTool` 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.

2.2.1 Prerequisites

Before the toolchain of the `invTool` can be used, the tool needs a *SPICE* / Verilog description of the circuit and the timing file (**.sdf*) for the circuit. These files can be generated using for example Cadence Encounter. More information on how to prepare a new circuit in Section 4.1.

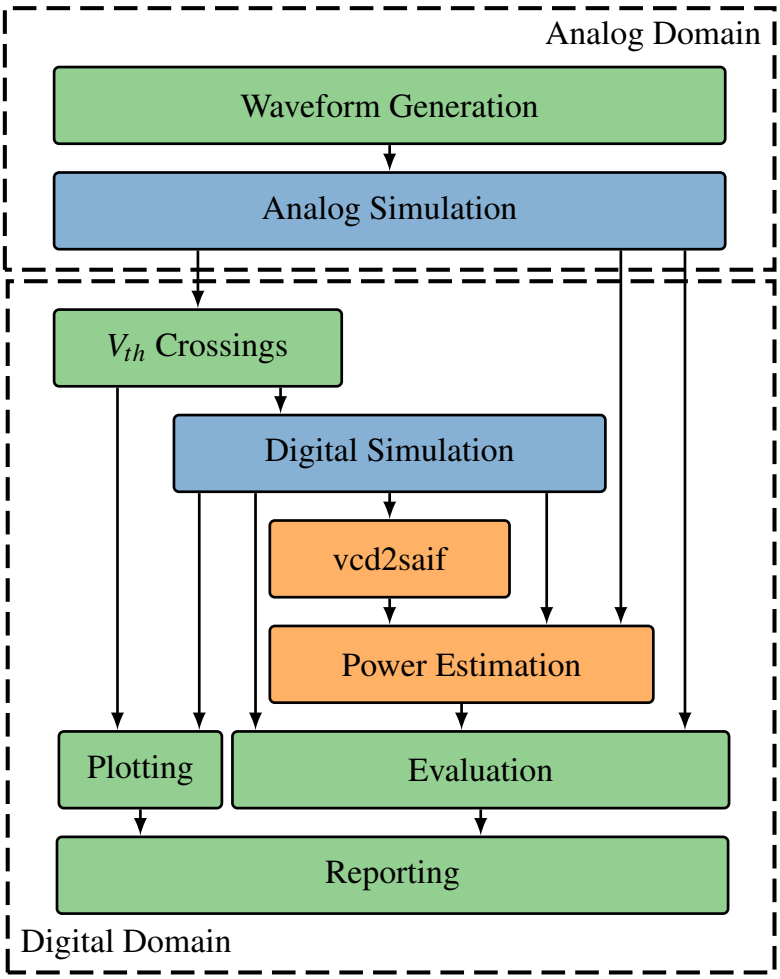


Figure 1: The workflow of the `invTool`.

2.2.2 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 3.1. This section also prepares the **.sp* file by replacing placeholders and inserting the generated waveform. This file is later used by *HSPICE*, *Spectre* or any other *SPICE* simulation tool.

2.2.3 Analog Simulation

Makefile recipes: *spice*

Uses the previously generated **.sp* file and runs the simulation with the specified *SPICE* simulation tool. The tool can be specified in the *config_xxnm.cfg* file by setting the variable ANALOG_SIMULATION_TOOL to SPECTRE or HSPICE. These are currently the tools that are supported by the toolchain. Of course, the variables can also be set in the circuit specific *config.cfg* file as well, but the default is that *HSPICE* is used for 65 nm and *Spectre* is used for 15 nm.

2.2.4 Crossings

Makefile recipes: *crossings*

The result of this step is a *crossings.json* file, which contains the extracted and digitized information of the *SPICE* simulation. Depending on the used tool in the previous step, the following tasks are performed:

- *HSPICE*: The resulting **.tr0* file is parsed and digitized. This is rather time-consuming, since this trace files can get very large. For future releases, parsing the **.tr0* should be replaced with converting the value-change-dump (**.vcd*) file. This is more efficient, since the required data is already digitized, and therefore the file size is reduced.

- *Spectre*: The digitized version of the trace is already generated in the previous step, by extracting the trace from binary trace file (**.raw*). In this step, the resulting **.json* file is only copied into the crossings folder.

2.2.5 ModelSim

Makefile recipes: *read, gates, sim*

Based on the *crossings.json* file, vector files are generated for each input, which are later applied to the circuit through the testbench. Since simple gates can automatically be generated (see Section 3.2), these gates also need to be created before the *ModelSim* simulation starts. The actual simulation uses **.do* files which have been prepared in previous steps, and generate **.vcd* files.

2.2.6 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 switching activity file (**.saif*) file, which 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.

2.2.7 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 *SPICE*. This step also calculates the deviation between the *SPICE* 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 all resulting rows are 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.

3 Configuration

3.1 Waveform generation

Listing 1 shows an example configuration for the waveform generation:

```
{
  "N": 500,
  "calc_next_transition_mode": "GLOBAL",
  "mue": 0.029,
  "sigma": 0.01,
  "rise_time": 0.002,
  "signals": [
    "nx1",
    "nx7",
    "nx3",
    "nx2",
    "nx6"
  ],
  "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"
      ]
    }
  ]
}
```

Listing 1: Example configuration for waveform generation.

3.1.1 Parameters

- N: the overall number of input transitions to generate. Default value: 100 transitions.
- 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. This is the default value.
- mue / sigma: Used for parametrizing the normal distribution, which is responsible for generation of the time for the next transition (delta time since the last transition). The values are in ns, default values: 0.029 ns / 0.010 ns.
- signals: A list of signals for which input waveforms should be generated.
- rise_time: Specifies the rise and fall time of the generated transitions in ns. Default value: 0.001 ns. Increasing this value can be useful if the *SPICE* simulation tool encounters numerical issues.
- 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. Values are in ns, default values: 0.010 ns / 0.030 ns.
- 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. Default value: 0.5.
- 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. Default value: False.

3.2 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 overridden by a file with the same name which is placed in the top level of the specific circuit.

3.2.1 Configuration

Listing 2 shows a basic configuration for a NAND gate with two inputs and one output:

```
"ND2M1N": {
  "T_P": 1,
  "channel_type" : "HILL_CHANNEL",
  "channel_location" : "INPUT",
  "channel_parameters" : {
    "N_UP" : 1.0,
    "N_DO" : 1.0
  },
  "entity_name": "ND2M1N",
  "function": "nand",
  "inputs": [
    "A",
    "B"
  ],
  "outputs": [
    "Z"
  ]
}
```

Listing 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 involution channel.
- channel_type: Currently HILL_CHANNEL and EXP_CHANNEL are supported. The EXP_CHANNEL is the default setting.
- channel_location: The following options are available: INPUT, OUTPUT, OUTPUT_SWAPPED. Either the involution 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.

- channel_parameters: Depending on the used channel, specific parameters can be specified here, which are then passed to the channel implementation. Currently, only N_UP and N_DO are supported by the Hill-channel. The exp-channel requires no additional channel parameters.
- 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.

3.2.2 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. In order to use these gates with the multi execution tool, the implementation needs to offer an architecture for each simulated combination of channel_location and channel_type. Currently, two channel types and three channel locations are supported, therefore the gates have to offer six implementations (if all combinations should be simulated). Note that the switch between the architectures is not working yet (as described in Section 7). An example of a manually implemented gate can be found in experiment_setup/vhdl/gates/ND2N1N.vhd.

3.3 Report generation

The report generation of the `invTool` 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*, *Spectre*, *Design Compiler*, *PrimeTime*).
- LaTeX templates for displaying the data in a customer-defined way.

- Scripts for generating **.tex* 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

3.3.1 Scripts for extracting data

With various scripts, the data of the simulation is extracted from the reports of the tools (*HSPICE*, *Spectre*, *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 `invTool`. 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 *variables.tex*, which is included by the top level report file. All the specified variables can be used in the report files.

3.3.2 Latex templates

The `invTool` 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 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 3.1). *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. %##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.

3.3.3 Report config file

The *report.cfg* shown in Listing 3 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 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 4.2.

3.3.4 Plots

During the report generation a number of plots is generated, based on the result traces of *SPICE* and *ModelSim*. In the *config.cfg* (either in the general, or in the circuit specific config file) a number of parameters can be set. An example is shown in Listing 4:

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

Listing 4: Figure plotting configuration.

- 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, shown in Listing 5 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.

```
1 export FIGURE_INV_EXPORT_DEV_TRACE_INFO=True
2 export FIGURE_MSIM_EXPORT_DEV_TRACE_INFO=True
```

Listing 5: Deviation trace export configuration.

4 How-To

4.1 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:

4.1.1 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/ *.spf*: file containing the circuit under test, defined as a subcircuit, which is used in *main_new.sp*. Note that for some ciruits a **.spf* file is used. Nevertheless, in the following sections a *SPICE* file has always the file extension **.sp*.

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

  - <VDD>
  - <VTH>
  - <TEMP>: Can be used to set the temperature for the simulation.
  - <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 recommended 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 can be very steep (depending on the configured rise_time), 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 6.

```
1 * circuit: inv tree
2 .LIB <SPICE_LIB>
3 .INCLUDE <SPICE_CIR>
4
5 * main circuit
6 .INCLUDE ../inv_tree.sp
7
8 * 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 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 a **.vhd* file can be used to specify the circuit (see *config.cfg* in the circuit directory). 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. Note that the sdf-extract tool in Section 5.3 creates **.sdf* files with INTERCONNECT = 0, since the channel model is not able to incorporate INTERCONNECTs yet. Therefore the **.v* file can be used without problems, and the warnings during the *ModelSim* simulation can be ignored.

- **.sdf*: Contains timing information, required for both delay models (*ModelSim*, Involution). It contains information about the timing re-

garding INTERCONNECTs and also about cell internal timing.

- **.spef*: This file is optional, but it is highly recommended to add a standard parasitics exchange format (**.spef*) file to the circuit. This file can be for example created during the design process with Cadence Encounter, and is used during the power estimation with *Design Compiler* and *PrimeTime*. Adding such a file increases the accuracy of the power estimation in general.

4.1.2 Configuration files

- Makefile: The Makefile includes the variables from the circuit configuration 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 which cannot be done with the default set of functions provided by the `invTool`.

- *generate.json*, see Section 3.1

- *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 in Listing 7 contains an example *matching* file:

```
1  din din
2  xmycir.g10:z temp1
3  xmycir.g11:z temp2
4  xmycir.g12:z temp3
5  xmycir.g13:z temp4
6  xmycir.g14:z temp5
7  xmycir.g15:z temp51
8  xmycir.g16:z temp52
9  xmycir.g17:z dout1
10 xmycir.g18:z dout2
11 xmycir.g19:z dout3
12 xmycir.g20:z dout4
```

Listing 7: Example configuration for matching file.

Note that there exists a tool for generating the *matching* file, see Section 5.4. The best way to generate the matching is by matching the outputs of the gates between **.sp* and **.v* file.

- *report.cfg*: Contains configuration for the automatic report generation, see Section 3.3.
- *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 in Listing 8 shows an example configuration:

```
export TOP_DIR:=$(dir $(abspath $(lastword
↪  $(MAKEFILE_LIST))))
export INPUT_NAMES=din
ifndef MULTI_EXEC
include ../config.cfg
include ../config_65nm.cfg
else
include ${ME_CIRCUIT_DIR}/config.cfg
include ${ME_CIRCUIT_DIR}/config_65nm.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 CIRCUIT_FILE_TYPE=verilog
export CIRCUIT_FILE=inv_tree_post.vh
export VERILOG_FILE=inv_tree_post.vh
export VCD_SIGNALS=din temp* dout*
export SDF_FILE=inv_tree_30.sdf
export REQUIRED_GATES=CKINVM1N
export SPEF_FILE_NAME=inv_tree.spef
```

Listing 8: Example configuration for circuit configuration file.

Specifying the TOP_DIR is required for most of the other path variables, since 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 INPUT_NAMES contain the names of the inputs of the Verilog circuit. This variable has to be defined before including the other configuration files, since they make use of this variable. 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 5.1). The circuit specific file always includes the general configuration file and the configuration file for the used technology first. Afterwards, variables set in these files can be overridden, if they should be set specifically for a circuit. 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*, *Design Compiler* and *PrimeTime*. REQUIRED_GATES contains a list of gates which are used by this circuit, and should be auto generated, more information in Section 3.2. SPEF_FILE_NAME is optional, and can be used to add a standard parasitic exchange format (**.spef*) file which is used during power estimation with *Design Compiler* and *PrimeTime*. CIRCUIT_FILE_TYPE can be either verilog or vhdl. In the first case, the CIRCUIT_FILE and the VERILOG_FILE are the same, in the latter case, CIRCUIT_FILE is set to the *.vhd* file. This can be required when the INTERCONNECT warnings at the outputs of the circuit should be avoided.

- *schematic.png*: Optional, is used for reporting (see Section 4.2).

4.2 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)

5 Tools

5.1 Multi execution tool (multi_exec)

The multi_exec 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, channel_location, channel specific parameters). The tool also allows to simulate the same circuit multiple times with the same configuration but with different randomly generated waveforms.

5.1.1 Configuration

The configuration of the multi_exec tool is contained in two different files that can be seen in Listing 9 and 10

```
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 9: Configuration for the multi_exec 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.

```
{
  "N": 2,
  "keep_waveform" : true,
  "gate_generation": {
    "t_p_list": [
      0.8,
      1.2
    ],
    "channel_location_list": [
      "INPUT",
      "OUTPUT"
    ],
    "channel_type_list": [
      {
        "channel_type" : "EXP_CHANNEL",
        "channel_parameters" : {
        }
      },
      {
        "channel_type" : "HILL_CHANNEL",
        "channel_parameters" : {
          "N_UP" : 1.0,
          "N_DO" : 1.0
        }
      }
    ],
  },
  "waveform_generation": [
    {
      "calc_next_transition_mode": "GLOBAL",
      "groups": [
```

```
      ],
      "mue": 0.029,
      "sigma": 0.01
    },
    {
      "calc_next_transition_mode": "LOCAL",
      "mue": 0.029,
      "sigma": 0.02
    }
  ]
}
```

Listing 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, channel_location_list), 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 (*SPICE* Simulation, *ModelSim* delay model simulation) is not re-executed, as it leads to the same result as the previous execution.
- gate_generation: channel_location_list contains the different channel locations over which should be swept, (t_p_list) contains the values for the pure delay T_P in ns. These settings are used for all gates, so if multiple kinds of gates are used in the circuit, all gates get the same parameters. It is also possible to sweep over different channel types (channel_type_list) and different channel parameters.
- waveform_generation: Contains a list of waveform generation configurations (see Section 3.1). If parameters are not specified here, the value

of the circuit configuration file is used as fallback value.

5.1.2 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 report 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.jsonnum* and *gate_config.jsonnum*. 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.

5.1.3 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* files. The configuration for the reporting is made in the *multi_exec.cfg* file as shown in Listing 11

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

Listing 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 shown in Listing 12

```
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 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. If this property is 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 (*SPICE* / *SPICE*) and to the result of the simulation of the *SPICE* trace with the same tool (*SPICE* / 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.

5.2 Plots

For plotting the results of the multi_exec tool a Python script is used, which uses matplotlib to display the results. In order to work with the script the *results.csv* file has to be converted into an *evaluation.csv* file, where only the aggregated results are contained.

Currently, this conversion step is performed manually. The steps which have to be made are described in the *README* file. Of course, future versions of the plotting script should be able to automate this process.

Since more than 20 metrics are supported, and result files can easily contain more than 20 different configurations, the script offers the possibility to filter for certain metrics and configurations. Otherwise, the plots would be too overloaded. For information on how to filter for specific metrics and configurations, please refer to the *README*.

5.3 SDF Extract

During the evaluation of the sample circuits, inaccuracies in the **.sdf* files have been encountered. Therefore Python scripts have been created which can create custom **.sdf* files. Moreover, the tool can not incorporate INTERCONNECT delays yet, and therefore **.sdf* files containing such delay need to be adapted.

The process consists of two scripts:

1. *generateDependencies.py*: The *dependency tree* of the circuit has to be extracted. This is done by reading the standard **.sdf* file, and extracting all INTERCONNECTs. This yields all output to input connections.

Based on these results the output to output connections are generated. These results are stored in a dictionary, where for each gate instance the name of the previous output and its own output are stored.

2. *extractSdf.py*: In the second step the result (**.vcd* file) of a *SPICE* simulation is used to generate the custom **.sdf* file. The simulation should have sufficient transitions with long pauses during each transition so that for each gate instance and each input the falling and rising time can be extracted from the **.vcd* file. If there are more delay values for one value in the **.sdf*, the average of these delay values is used.

   The final step of the script is that the values in the standard **.sdf* file are overwritten with the generated values.

Requirements:

- standard **.sdf* file.
- **.vcd* file which contains the *SPICE* simulation results.

Note that the delay for an instance is always calculated from the output of the previous instance to the output of this instance. Of course, one could also calculate the delay from the input of the current gate to the next gate, but then there are multiple possibilities, since the gate could drive several other gates.

Note that the scripts are tailored for the sample circuits, and that they need to be generalized so that they can be used for all circuits.

5.4 Generate Matching

Generating the matching for the *mips_clock_15nm* circuit was a very tedious and error prone task. Therefore we created a Python script which extracts the matching between the **.sp* and **.v* file.

The script extracts all gate outputs from the **.sp* and tries to find the corresponding outputs in the **.v*.

Note that this script is only working for the *mips_clock_15nm* circuit, and that implementing a general script for generating the matching is a very demanding task.

6 Evaluation

There are several metrics that can be used for comparison of the different delay models and parameters. They can be divided into two types: Power comparison and waveform comparison. In the following, both types are described.

6.1 Power comparison

The *SPICE* simulation performs two different power estimations - the average power consumption and the maximum power consumption.

6.1.1 Average power consumption

For the average based power consumption, two types of reference values are used. The first one is the estimation by the *SPICE* simulation (*SPICE*/*SPICE*), and the second one is the power estimation by *Design Compiler* and *PrimeTime* based on the trace of *SPICE* simulation (*SPICE*/column). Using these two types of reference values is helpful for distinguishing deviations because of different tools and deviations because of different delay models. In general, the two types of reference values should be similar ($\pm 5\,\%$). Using a **.spef* file for the power estimation with *Design Compiler* and *PrimeTime* increases the accuracy of these tools, and therefore in general reduces the deviation to the *SPICE* reference.

6.1.2 Maximum power consumption

The time-based power estimation method of *PrimeTime* can also estimate the maximum power consumption. This result can be compared to the estimation of the *SPICE* simulation, but unfortunately the results can differ significantly, and therefore this metric was discarded.

6.2 Waveform comparison

During the reporting process, the resulting traces of the *ModelSim* simulation with the standard delay model and the involution delay model are compared to the *SPICE* trace, which is used as reference. In the following sections, the different metrics that are based on the waveforms are described.

6.2.1 Number of transitions

One of the simplest metrics is to sum up the number of transitions on each trace and compare them. The `invTool` also calculates the maximum deviation on a single signal, in order to find out the part of the circuit, where the deviation is the most.

6.2.2 Area under the deviation trace

For the deviation trace, the absolute value of the difference between a trace (*ModelSim* or Involution delay model) and the *SPICE* reference trace is calculated. The area under the deviation trace is calculated by multiplying the duration of the deviation with the value of the deviation (V_{DD}). A more sophisticated approach is to find corresponding transitions on both compared traces, and calculated the signed area of the deviation. The area is counted positive if the compared trace is leading against the *SPICE* trace, otherwise the area is counted negative. This metric can help to find problems in the **.sdf*, for example if the delays are to long in general, the negative area is significantly larger than the positive area.

6.2.3 Glitches

The `invTool` also counts the number of glitches. Several types can be distinguished:

- Type of glitch:
  - original (o): This is what is in general considered as typical glitch. Both signals are on the same level, then two subsequent transitions happen on one of the two signals.
  - inverted (i): Both signals are one opposite levels, then two subsequent transitions happen on one signal, and after these two transitions they are again on different levels.
- Location of glitch:
  - induced (i): If the glitch is on the trace of the digital delay model. Compared to the *SPICE* reference, a glitch has been induced.

– suppressed (s): Two subsequent transitions happen on the *SPICE* reference signal, and these two transitions are not made by the simulation with the digital delay model.

Two subsequent transitions on one trace without a transition on the other trace are called glitch. Figure 2 shows all four combinations of glitches according to the classification above. The first character denotes the type of the glitch, the second the location of the glitch. Figure 2 also shows how the deviation trace for two arbitrary traces looks like.

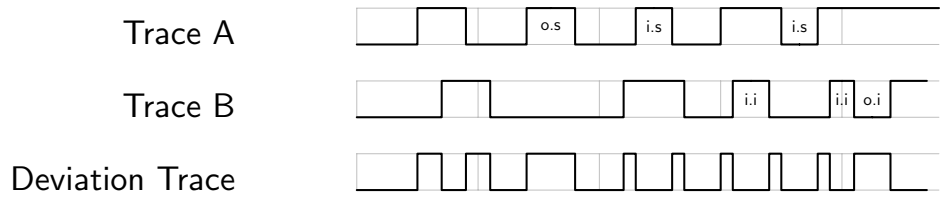


Figure 2: Types of glitches.

Note that for better comparability between different circuits, the number of glitches can be divided by overall number of transitions. We decided to use for the induced glitches the number of transitions on the corresponding *ModelSim* / Involution delay model trace, and for the suppressed glitches the number of transitions on the reference signal. This process is currently performed manually on the resulting *results.csv* file of the multi_exec tool. The plotting tool (Section 5.2) already supports these glitch percentage values.

6.2.4 Leading / Trailing against reference

The leading / trailing metrics aim at mapping the area under deviation and the number of transitions to an average value how soon or late transitions take place, compared to the *SPICE* reference. Note that these metrics are not directly calculated by the `invTool`, but all the required values are extracted by the tool and therefore they can be easily calculated during the post processing. The plotting tool described in Section 5.2 already supports these metrics. Nevertheless the preparation for the plotting tool is still performed manually (as described in Section 5.2).

They come in three different flavors:

- the positive / negative area under the deviation trace divided by the number of transitions on the *SPICE* reference.

- the positive / negative area under the deviation trace divided by the number of transitions where the trace was leading / trailing compared to the *SPICE* reference.
- same as previous, but only transitions which are no glitch are considered.

6.2.5 Remarks

Note that the described metrics are by far not complete, and that these are only the ones we considered most important during the development process. Due to the structure of the `invTool`, extracting more values and adding new metrics should be easy. Moreover, for different task other metrics might be more suitable.

7 Future development

During the development and especially during the usage of the `invTool` we came up with ideas for useful improvements. The following is a list of open tasks, each with a short description, which would further increase the usability and / or the performance

- When *HSPICE* is used, the *crossings.json* file is generated from the **.tr0* file. *HSPICE* automatically supports the generation of a **.vcd* file, which is already the digitized version of the trace. Using the **.vcd* for the generation of the *crossings.json* would increase the performance. (PERFORMANCE).
- Manually implemented gates cannot be used in combination with the multi_exec tool. The idea is to implement different architectures for each combination of channel type and channel location and set the different architectures during the simulations. The configuration for setting the different architectures is not working yet. This has to be implemented as soon as manually implemented gates are used. (FEATURE).
- When using multiple gates, the report generation for the gate configuration section is not working yet. The configuration is only displayed for one gate. (USABILITY).

- In *makeVCD.py* the symbol pool is currently fixed to a specific amount of signals. For larger circuits, this has to be adapted to generate allowed symbol out of a pool of valid characters. (FEATURES).
- The tool uses Python2.6 and Python3.6 (only for the *make.vcd*, which uses *rawread.py*). Either change *rawread.py* so that it can be run by Python2.6, or update all the other scripts so that they are running with Python3.6 (preferred). (USABILITY).
- The name of the Makefile recipe for the simulation with the digital delay model is currently named *modelsim*. Since the `invTool` is not only limited to *ModelSim*as digital simulation tool, this recipe and the output folder should be renamed to something more general (*make digital*?). (CONSISTENCY).
- Rethink the naming conventions for the extracted values out of the different files of the tools. Especially important if different tools than the standard tools are used. (USABILITY).
- The tools described in Section 5.3 and Section 5.4 need to be generalized, so that they work for multiple circuits. (FEATURE).

7.1 Implemented Improvements

- V2.1: The file (*evaluation.csv*) that is used by the plotting tool (Section 5.2) is currently prepared manually out of the *results.csv*. This process should be automated in the future with a Python script (group data, add additional information, calculate metrics). (FEATURE). The script *prepareData.py* takes now care of automatically preparing the data for plotting.

8 Documentation TODOs

- Describe *generateGateLibrary.py* script from Jürgen
- Describe *evaluation* folder and the newly added script *prepareData.py* which automatically generates the input file for plotting.
- Describe the T_p percent mode, which allows gate instance specific pure delays, based on δ_∞.

References

[1] 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